(19) KOREAN INTELLECTUAL PROPERTY OFFICE

KOREAN PATENT ABSTRACTS

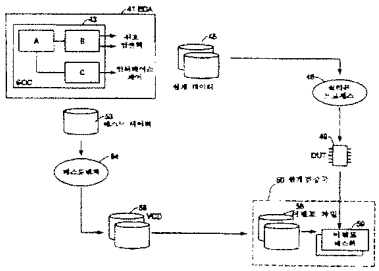
(11)Publication number: 1020010067370 A
(43)Date of publication of application: 12.07.2001

(21)Application number: 1020000063483
(22)Date of filing: 27.10.2000
(30)Priority: 28.10.1999 US1999 428746
(51)Int. Cl G06F 11/26

(71)Applicant: ADVANTEST CORPORATION
(72)Inventor: RAJSUMAN ROCHIT
YAMOTO HIROAKI

(54) METHOD AND APPARATUS FOR SOC DESIGN VALIDATION

(57) Abstract:



PURPOSE: A method and an apparatus for SoC design validation are provided to verify the design of a system on chip accurately at high speed and at a low cost. CONSTITUTION: The method comprises a step for verifying each core integrated in a system on a system on chip, a step using a silicon a silicon IC corresponding to each core and a simulation test bench from a core provider, a step for verifying the interface between respective cores, the core, an on chip bus and a glue logic using an FPGA or emulation of a test bench and the glue logic, a step for verifying the timing between cores and the timing of system on chip level, and a step for verifying design of the entire system by using the simulation test benches of the entire system on chip and executing applications.

Legal Status
Date of request for an examination (20021024)
Notification date of refusal decision ( )
Final disposal of an application (registration)
Date of final disposal of an application (20050318)
Patent registration number (1004914610000)
Date of registration (20050517)

Number of opposition against the grant of a patent ( )
Date of opposition against the grant of a patent ( )
Number of trial against decision to refuse ( )
Date of requesting trial against decision to refuse ( )

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl. [7]
G06F 11/26

(11) 공개번호 특2001-0067370
(43) 공개일자 2001년07월12일

(21) 출원번호 10-2000-0063483
(22) 출원일자 2000년10월27일

(30) 우선권주장 09/428,746 1999년10월28일 미국(US)

(71) 출원인 가부시키가이샤 어드밴티스트
일본국 도쿄도 네리마구 아사히쵸 1쵸메 32반1고

(72) 발명자 라즈스만록히트
미국95054캘리포니아주산타클라라스코트불러바드3201
야모또히로아끼
미국95054캘리포니아주산타클라라스코트불러바드3201

(74) 대리인 장수길
구영창

심사청구 : 없음

(54) SoC 설계 검증을 위한 방법 및 장치

요약

본 발명은 정확성이 높고 고속이며 비용이 저렴한 SoC(system-on-a-chip) 설계 검증을 위한 방법 및 장치에 관한 것이다. 상기 장치는, 코어 제공자(core provider)에 의해 제공된 각각의 코어의 실리콘 IC 및 시뮬레이션 테스트벤치들 (simulation testbenches)을 사용하여, SoC에 집적될 개별 코어들을 검증하는 단계; SoC 설계자에 의해 개발된 시뮬레이션 테스트벤치들과 글루 로직(glue logic)의 FPGA/에뮬레이션(emulation)을 사용하여 개별 코어들 간의 인터페이스, 코어들의 온-칩 버스들(on-chip buses), 및 글루 로직을 검증하는 단계; 코어-투-코어 타이밍(core-to-core timings)과 SoC 레벨 타이밍 임계 경로들을 검증하는 단계; 및 전체 SoC의 시뮬레이션 테스트벤치 및 어플리케이션 실행(application runs)을 사용하여 전체 설계 검증을 실행하는 단계를 포함하는 방법을 사용하도록 허용한다
.

대표도
도 3

색인어

시스템-온-칩(SoC), 실리콘 IC, 제어 CPU, 메인 시스템 컴퓨터, 설계 검증국, 검증 유닛, 시스템 버스

명세서

도면의 간단한 설명

도 1은 기능 코어들의 설계에서의 추상 레벨들과 관련 검증 방법들을 도시한 도면.

도 2는 본 발명에 따른 시스템-온-칩(SoC; system-on-a chip) IC의 전체 개념 및 설계 검증 절차를 도시한 순서도.

도 3은 전자 설계 자동화(EDA; electonic design automation) 환경과 본 발명의 설계 검증국 간의 관계를 포함하는 본 발명의 설계 검증의 전체 개념을 도시한 개략 블록도.

도 4a는 도 3의 EDA 환경에서 설계된 SoC의 일례를 도시한 블록도.

도 4b는 도 3의 설계 검증국의 기본 구성의 일례를 도시한 블록도.

도 5는 다수의 검증 유닛들(VU; verification units)이 제공되는 본 발명의 설계 검증국의 구성의 일례를 보다 상세히 도시한 블록도.

도 6은 한 형태의 이벤트 테스터(event tester)를 갖는 도 5의 검증 유닛(VU)의 구성의 일례를 도시한 블록도.

도 7은 테스트 중 SoC의 글루 로직을 평가하기 위한 검증 유닛의 구조의 일례를 도시한 블록도.

도 8은 테스트 중 SoC의 글루 로직을 평가하기 위한 검증 유닛의 구조의 다른 일례를 도시한 블록도.

도 9는 하이 레벨 어플리케이션 언어들을 지지하는 본 발명의 설계 검증국의 다른 일례를 도시한 블록도.

도 10은 테스트될 장착 기능 코어들과 성능 보드(performance board)와의 구조 관계를 도시하는 본 발명의 설계 검증국의 또 다른 일례를 도시한 블록도.

도 11은 다수의 검증 유닛(VU)들이 메인 시스템 컴퓨터에 의해 직접 제어되는 본 발명의 설계 검증국의 또 다른 일례를 도시한 블록도.

< 도면의 주요 부분에 대한 부호의 설명 >

41 : EDA(전자 설계 자동화)

43 : 시스템-온-칩(SoC)

45 : 설계 데이터 파일

62 : 메인 시스템 CPU

64 : 시스템 버스

66 : 검증 유닛

67 : 제어 CPU

68 : 실리콘 IC

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 다수의 기능 코어들을 갖는 SoC(시스템-온-칩) IC의 설계 완전성 (design integrity)을 검사하기 위한 방법 및 장치에 관한 것으로, 특히, 설계 검증이 코어 각각의 의도된 기능, 각각의 코어의 타이밍, 코어들 간의 인터페이스, 및 SoC IC 전체 시스템 동작에 대하여 평가될 수 있는 SoC 설계 검증을 위한 방법 및 장치에 관한 것이다.

최근 5년 동안, ASIC 기술은 칩-세트(chip-set) 개념으로부터 시작해서 내장 코어 기저 시스템-온-칩(SoC) 개념까지 발전되어 왔다. 이러한 시스템-칩들은 다양한 어플리케이션들을 제공하는 " 코어들" 로 공지된(또한 지적 재산 또는 IP로 공지된) 복합 기능의 미리 설계된 모델들을 사용하여 조립된다. 상기 코어들은 일반적으로 베릴로그/HDL(Verilog/HDL; 소프트-코어로 공지됨)과 같은 하이-레벨 기술 언어(HDL; high-level description language)에서 또는 GDSII(하드-코어로 공지됨)와 같은 트랜지스터 레벨 레이아웃에서 유용하다. 시스템-칩은 마이크로프로세서, 대형 메모리 배열들, 오디오 및 비디오 제어기들, 모뎀, 인터넷 튜너, 2D 및 3D 그래픽 제어기들, DSP 기능들(functions) 등과 같은 온-칩 기능들을 위한 하드 코어 및 소프트 코어의 결합들을 포함할 수 있다.

상기 코어들은 코어 제공자 회사들로부터 수회 취득되었고 SoC를 만들기 위해 함께 집적되었다. 코어가 외부로부터 취득되면, 코어 제공자는 코어의 시뮬레이션 테스트벤치와 함께 설계 네트리스트(netlist)를 제공한다. 따라서, 코어가 SoC에 집적되는 경우, 집적 SoC 설계에서의 동작을 검증하기 위해 임의의 변형 없이 코어의 테스트벤치를 직접 적용하는 것이 바람직하다.

설계는 베릴로그/VHDL과 같은 하이-레벨 언어를 사용하는 블록들 및 서브-블록들로 기술되고 행동/게이트-레벨 베릴로그/VHDL 시뮬레이터에 의해 시뮬레이트된다. 그러한 시뮬레이션은 설계가 실리콘 IC로 제조되기 이전에 기능을 검사하기 위한 것을 목표로 한다. 설계 검증은 전체 기능을 검증하지 않고는 설계 에러들이 발견되지도 제거되지도 않기 때문에 가장 중요하고 어려운 작업들(tasks) 중의 하나이다. 시뮬레이션의 속도가 느리고 SoC 설계 사이즈가 크기 때문에, SoC 레벨 설계 검증은 오늘날의 도구와 방법으로는 거의 불가능한 작업이다.

검증은 용인된 실체(accepted entity)에 대한 검사를 함축한다. 이는 시스템 설계에 있어서 명세서를 배경으로 설계를 검사하는 것을 의미한다. 검증은 시스템 설계 중에 하나의 추상 레벨로부터 다른 추상 레벨로의 변환이 정확하게 이루어졌는지를 검증하는 것이 아니다. 검증의 목적은 실제의 한계들 내에서 시스템이 구현되고 제조된 후에 의도대로 작동할 것인지를 알아내는 데 있다. 시스템-온-칩은 다수의 내장 코어들을 갖는 단일 하드웨어 구성 요소를 포함한다. 따라서, SoC의 설계 검증은 코어들을 검증하고, 코어들 간의 상호 접속들을 검증하고, 결합된 시스템 동작을 검증하는 것으로 이루어진다.

오늘날, SoC 명세서의 개발과 함께 행동 모델들이 개발됨으로써, 시뮬레이션 테스트벤치가 설계 검증 또는 시스템 동작의 검증을 위해 생성될 수 있다. 시스템 레벨 검증은 설계 계층 구조(design hierarchy)를 근거로 이루어진다. 먼저, 일반적으로 코어 레벨에서 리프(leaf) 레벨 블록들이 스탠드 얼론 방식(stand-alone way)으로 정확성에 대한 검사를 받는다. 그 후, 코어들 간의 인터페이스들이 트랜잭션 타입과 데이터 내용면에서 정확성에 대한 검사를 받는다. 다음 단계는 전체적으로 어셈블된 칩(fully assembled chip)에서 어플리케이션 소프트웨어 또는 등가 테스트벤치를 실행하

는 단계이다. 이는 하드웨어/소프트웨어 공동 검증을 포함한다 (1998년의 Kluwer Academic Press에 의한 M. Keating과 P. Bricaud의 " 재사용 방법 수동(Reuse methodology manual)" ; 1997년의 Kluwer Academic Press에 의한 J. Stcunstrub과 W. Wolf의 " 하드웨어-소프트웨어 공동 설계(Hardware-software co-design)"). 소프트웨어는 소프트웨어 코드의 실행 시간동안의 실행들에 의해서만 검증될 수 있으므로, 하드웨어-소프트웨어 공동-시뮬레이션(co-simulation)이 실행될 수 있다. 또한 종종 ASIC(응용 주문형 집적회로; application specific IC) 형태 또는 FPGA(현장 프로그램가능한 게이트 어레이; field programmable gate array)를 사용한 하드웨어 기본형(prototype)이 개발되어 전체 시스템 동작의 검증을 위해 사용된다.

기능 검증(Functional Verification)

도 1은 상이한 추상 레벨에서의 코어 설계 및 각 레벨에서 오늘날 어떤 타입의 검증 방법이 사용되는지를 도시한 것이다. 도 1은 최저 추상 레벨로부터 최고 추상 레벨까지 도시하고 있다. 즉, 도 1은 행동 HDL 레벨(21), RTL(register transfer language) 레벨(23), 게이트 레벨(25) 및 물리 설계 레벨(physical design level; 27)을 도시하고 있다. 상이한 추상 레벨들에 대응하는 검증 방법들은 도 1의 블록(28)에 리스트되어 있다. 기본 타입의 검증 테스트들은 다음을 포함할 수 있다.

(1) 설계 명세서에 순응하는지를 확인하기 위한 순응 테스팅(compliance testing).

(2) 전압, 온도 및 공정에 있어서 최소 상태이거나 최대 상태인 복합 시나리오 및 코너 케이스에 대한 코너 테스팅.

(3) 상당히 불명료한 버그들을 검출할 수 있는 테스팅으로 기본적으로 목표가 되지 않은(non-targeted) 랜덤 테스팅.

(4) 기능에 있어서의 임의의 오류(misrepresentation)가 정정될 수 있도록 설계상의 실제 어플리케이션을 실행함으로써 이루어지는 실제 코드 테스팅(real code testing).

(5) 설계에 있어서 임의의 변형 후에 테스트 집합(collection)을 실행함으로써 행해지는 회귀 테스팅(regression testing). 각각의 버그 픽스(bug fix)는 추가 테스트 조건들을 갖는 새로운 테스트 케이스를 추가할 것을 요구한다.

테스트벤치의 발달은 코어 및 타겟 SoC의 기능에 따른다. 예를 들면, 프로세서를 위한 테스트벤치는 명령 세트를 토대로 테스트 프로그램들을 실행하지만, PCI 코어와 같은 버스 제어기 인터페이스 코어에 대한 테스트벤치는 자극을 적용하고 시뮬레이션 출력 결과들을 검사하기 위해 버스 기능 모델들과 버스 모니터들을 사용한다. 이러한 기법의 문제점은 행동 테스트벤치들이 매우 느리다는 것이다.

테스트 케이스들(자극 또는 패턴)을 생성한 후에, 출력 응답들이 정확한지의 여부를 검사할 필요가 있다. 오늘날은 출력 파형을 관찰함으로써 수동으로 행해지지만, 설계가 변경됨에 따라 수동 검사가 불가능해진다. 출력 응답들을 검증하기 위한 다른 방법은 기본적으로 하드웨어/소프트웨어 공동-시뮬레이션인 실제 소프트웨어 어플리케이션을 실행하는 것이다. 이러한 접근은 오늘날의 계산 공급원들 (computational resources)에 있어서 매우 비효율적이다. 또한, 이와 같은 테스트벤치에서, 어플리케이션과 코어 간의 실제 트랜잭션들은 시뮬레이트될 총 사이클들 중 단지 작은 부분이다. 따라서, 기능의 단지 작은 부분만이 검증된다.

인터페이스 검증(Interface Verification)

SoC 설계에서, 코어들 간의 인터페이스들이 검증될 필요가 있다. 통상 인터페이스들은 코어들, 코어-투-코어 혹은 온-칩 글로벌 버스들을 접속하는 어드레스 및 데이터를 포함하는 정규 구조를 갖는다. 또한 인터페이스는 요구/승인 프로토콜 및 버스 제어기와 같은 몇몇 형태의 제어 메카니즘과 신호들을 갖는다. 이러한 인터페이스들의 정규 구조는 제한된 수의 데이터 시퀀스 및 제어 신호들의 트랜잭션에 의해 정의될 수 있다.

인터페이스 검증은 각각의 인터페이스에서 모든 가능한 트랜잭션들의 목록을 필요로하므로, 모든 가능한 테스트 케이스들이 생성될 수 없기 때문에 불가능한 작업이다. 따라서, 제한된 검증이 실행된다. 제한된 검증 후에, 다음 작업은 각각의 코어가 수신하는 데이터의 모든 값들과 모든 데이터 시퀀스들에 대해서 코어가 정확하게 행동하는지를 검증하는 것이다. 이 검증 역시 불가능하여, 트랜잭션의 모든 상이한 데이터 값들이 너무 크기 때문에 매우 불완전한 검증이 행해진다.

타이밍 검증(Timing Verification)

타이밍 검증은 기능 검증 보다 훨씬 더 어려운 작업이다. 정적 타이밍 분석은 오늘날 가장 흔하게 사용되는 방법이다. 정적 타이밍 분석은 다양한 기술 라이브러리들과 함께 합성된 코어들의 대표적인 네트리스트에 대해 실행된다. 정적 타이밍 분석은 거짓 경로들이 적합하게 여과되지 않기 때문에 비관적인 경향이 있다. 거짓 경로들을 제거하는 것은 수동 공정이며, 따라서 에러가 생기게 된다. 게이트-레벨 시뮬레이션은 이러한 타입의 에러들에 대한 적합한 검사를 제공하지만, 자극을 발전시키고 게이트-레벨의 모든 타이밍 경로들을 시뮬레이트하는 것은 과도한 시간이 걸리기 때문에 완전한 해결 방법이 될 수 없다. 또한, 최악의 케이스의 타이밍 시나리오들은 설계 엔지니어에 의해 적합하게 식별되기에는 너무 복잡하거나 수없이 많기 때문에 게이트-레벨 시뮬레이션에서 실행되지 않는다.

전체 SoC 설계 검증(Full SoC Design Validation)

SoC 설계 검증의 주요 목적은 최종 사용자(end user)에 의해 사용되는 방식으로 전체 시스템을 검증하는데 있다. 이는 모든 코어들의 전체 기능 모델들(즉, 하드웨어 모델들) 및 합당한 양의 실제 시스템 어플리케이션들을 필요로 한다. 새로운 시스템이면, 어플리케이션들이 존재하지 않을 수 있다. 주요 관건은 시뮬레이션 속도이다. 예를 들어, RTL(레지스터 전송 레벨)에서조차 프로세서의 동작 시스템을 부팅하는데 시간이 많이 걸린다. 시뮬레이션 속도를 향상시키기 위해 2개의 방법들 중 하나가 사용된다. 하나는 시뮬레이션을 보다 빨리 실행시키기 위한 보다 높은 레벨의 추상(abstractions)이고, 다른 하나는 하드웨어의 기본형 또는 에뮬레이션이다.

보다 높은 추상 모델들의 경우, 메모리 및 프로세서 코어들과 버스 기능 모델들 및 모니터들이 통신 블록들과 함께 트랜잭션들을 생성하고 검사하는데 사용되기 때문에, 기능 코어들, 행동 혹은 ISA(명령 세트 구조; instruction set architecture) 모델들용으로 RTL 모델들이 사용된다. 미디어 프로세서로서의 SoC에 있어서, 소정의 어플리케이션 코드는 시뮬레이션 환경에서 실행하도록 생성된다. 소프트웨어 어플리케이션들을 적용할 때는 거의 행해질 수 없는데, 실리콘이 작동하고 있는지 아니면 완전히 교착 상태(dead) 인지를 검사하고 기본 버그들을 찾는 것에 제한된다. 오늘날, 에러들은 통신 인터페이스의 버스 모니터 혹은 시퀀스 검사기/모니터를 사용하여 수동으로 검출된다. 그러나, 시뮬레이션 속도는 임의의 합당한 사이즈의 어플리케이션을 실행시키기에는 너무 느린 대략 초당 10 사이클이다.

하드웨어 및 소프트웨어가 동시 방식(concurrent way)으로 시뮬레이트될 때, 공동-시뮬레이션(co-simulation)이라 칭한다. 하드웨어는 C 언어 함수로 모델링될 수 있고, 전체 시스템은 단일 C 언어 프로그램 처럼 수행될 수 있다. 그러나, 그것은 구현 레벨(implementation level)에 있지 않으므로 설계 검증이 아니고, 오히려 행동 검증 혹은 타당성 조사(feasibility study)이다. HDL/RTL 기술은 하드웨어 구성요소의 구현을 나타내므로 전체 시스템 검증에 필요하다. 공동-시뮬레이션은 하나 이상의 HDL 스뮬레이터 혹은 C/C++ 프로그램(컴퓨터 작동 시스템으로부터의 컴파일러,

로더, 링커 및 다른 요소들을 필요로 하는) 들 간의 통신을 필요로 한다. 따라서, 공동-시뮬레이션에 있어서의 부가적인 문제는 다른 시뮬레이터들 간의 통신이다.

하드웨어 기본형(Hardware Prototyping)

모든 설계 팀들은 제1 실리콘이 전체적으로 기능하도록 시도했지만, 설계에서 처음으로 시스템으로 도입할 때 50% 이상이 실패한다. 이는 시스템 레벨 검증 또는 SoC 레벨 설계 검증의 부족으로 인한 것이다. 제1 시도를 신뢰성있게 성공시키기 위해서는, 보다 실제적인 어플리케이션들이 시뮬레이트되어야만 한다. 시뮬레이션 시간이 적합치 않게 길기 때문에, 오늘날로서는 상당히 비용이 많이 들더라도 실리콘 기본형들(prototypes)을 사용하는 방법밖에 없다. 유용한 기술들은 FPGA/ LPGA 및 에뮬레이션이다.

보다 소형의 설계를 위해, FPGA(field programmable gate array) 기본형 또는 LPGA(laser programmable gate array) 기본형이 제조될 수 있다. FPGA 및 LPGA는 비록 게이트 카운트 용량 및 ASIC의 속도가 부족하지만, 보다 소형의 블록들 또는 코어들에게는 양호한 반면, 전체 SoC에게는 적합하지 않다. 몇몇 FPGA들은 회로 보드에서 상호 접속될 수 있고 전체 SoC의 기본형을 만드는데 사용될 수 있다. 이러한 경우에, 버그 픽스가 SoC 칩의 재분할(re-partitioning)을 요구하면 FPGA들 간의 상호 접속들이 변형되어 새로운 회로 보드를 필요로 하게 되므로 변형하는데 비용과 시간이 상당히 많이 든다.

에뮬레이션 기술은 대안으로 대형 칩들을 위해 FPGA 집합을 제공한다. 프로그램 가능한 상호 접속, 고정 보드 설계들, 비교적 큰 게이트 카운트들과 특정 메모리 및 프로세서 서포트를 제공한다. 에뮬레이션은 전체 설계가 에뮬레이션 엔진 자체로 로드될 수 있는 경우 시뮬레이션 보다는 빠른 성능을 제공할 수 있다. 그러나, 실제 실리콘에 비해 실행 속도가 여전히 상당히 느리다. 칩 데이터 또는 테스트벤치 데이터의 상당한 부분이 호스트 컴퓨터에 로드되면 에뮬레이션 성능은 더 떨어진다. 상기 방법의 또 하나의 결점은 그것의 비용에 있다. 오늘날 상업적으로 유용한 모든 에뮬레이션 시스템들은 백만 달러가 넘는다.

설계가 너무 길 때(multi-million transistors; 멀티-밀리언 트랜지스터들), 실제 실리콘 기본형을 만들어서 최종 시스템을 디버그하는 것이 현재로서는 유일한 방법이다. 이러한 상황에서, 새로운 실리콘을 실행하지 않고 처음 몇 개의 버그들을 위해 동일한 실리콘이 사용될 수 있다. 전체 공정은 두세개의 실리콘 실행들(fabrication cycles; 제조 사이클들)을 필요로 하는데, 각각의 실행은 전체 제품 개발에 대하여 상당한 비용을 추가시킨다.

상술된 바와 같이, 현존 기술은 성능, 비용, 및 속도면에서 SoC 설계를 효율적으로 테스트하고 검증을 할 수 없다. 따라서, 고속으로 또한 저렴한 비용으로 SoC 설계 검증을 실행할 수 있는 새로운 방법 및 장치가 반도체 산업에서 필요하다.

발명이 이루고자 하는 기술적 과제

본 발명의 목적은 각각의 코어 기능, 코어들 간의 상호 접속, 및 전체 시스템 성능에 대한 SoC 설계 검증을 실행할 수 있는 SoC 설계 검증 방법 및 장치를 제공하는데 있다.

본 발명의 다른 목적은 빠른 속도와 저렴한 비용으로 SoC 설계 검증을 실행할 수 있는 SoC 설계 검증 방법 및 장치를 제공하는데 있다.

본 발명의 또 다른 목적은 SoC의 전체 기능을 검증하기 위해 SoC 설계 검증이 행해지는 설계 검증국(design validation station)을 제공하는데 있다.

본 발명의 또 다른 목적은 사용자가 현재 시스템들 보다 더 쉽게 SoC의 코어들의 오류를 디버그할 수 있게 하는 방법 및 장치를 제공하는데 있다.

본 발명은 내장 코어 기저 시스템-온-칩 IC(system-on-a-chip ICs)들의 설계 검증에 있어서 현재의 어려움들을 해결하는 새로운 설계 검증 또는 전체 기능 검증을 위한 방법 및 장치를 제공한다. 본 발명자는 상기 장치가 SoC의 전체 기능을 검증하는데 사용되기 때문에 설계 검증국이라고 한다. 본 명세서에 설명된 시스템 구조는 매우 효율적이고 비용이 적게 들며 임의의 이전 시스템과 기본적으로 상이하다.

본 발명의 한 양상은 다수의 기능 코어들이 집적되어 있는 내장 코어 기저 SoC(시스템-온-칩)들의 설계 검증 방법이다. 상기 방법은 코어 제공자에 의해 제공된 각각의 코어의 실리콘 IC 및 시뮬레이션 테스트벤치들을 사용하여, SoC에 집적될 개별 코어들을 검증하는 단계; SoC 설계자에 의해 개발된 시뮬레이션 테스트벤치들과 글루 로직의 FPGA/에뮬레이션을 사용하여 개별 코어들 간의 인터페이스, 코어들의 온-칩 버스들, 및 글루 로직을 검증하는 단계; 코어-투-코어 타이밍과 SoC 레벨 타이밍 임계 경로들을 검증하는 단계; 및 전체 SoC 및 어플리케이션 실행의 시뮬레이션 테스트벤치를 사용하여 전체 설계 검증을 실행하는 단계를 포함한다.

본 발명의 다른 양상은 SoC의 설계 검증을 위한 장치이다. 상기 장치는 사용자와 인터페이스하고 설계 검증을 위한 장치의 전체 동작을 제어하기 위한 메인 시스템 컴퓨터, 메인 시스템 컴퓨터로부터의 테스트벤치 데이터를 수신하고 SoC에 집적될 다수의 기능 코어들을 테스트하기 위해 테스트벤치 데이터를 사용하여 테스트 패턴들을 생성하는 다수의 검증 유닛들, 및 다수의 검증 유닛들과 메인 시스템 컴퓨터를 인터페이스하는 시스템 버스를 포함한다. 본 발명의 장치에 있어서, 다수의 실리콘 IC들은 검증 유닛들로부터의 테스트 패턴을 수신하기 위해 검증 유닛에 접속되고 검증 유닛과 메인 시스템 컴퓨터에 의해 평가될 응답 출력들을 생성한다. 실리콘 IC들은 SoC에 집적될 기능 코어들과 동일한 내부 구조 및 기능을 포함한다.

발명의 구성 및 작용

본 발명의 SoC 설계 검증의 전체 흐름은 도 2에 도시되어 있다. 검증 방법은 개별 코어들, 온-칩 버스와 글루 로직을 상호 접속하고, 타이밍을 검증하고, 최종적으로 SoC의 전체 시스템 성능을 검증하는 4 단계의 시스템 절차를 토대로 한다.

특히, 단계 S31에서, 검증 절차는 제일 먼저 실리콘 IC들 및 코어 기능 테스트벤치들을 사용하여 개별 코어들을 검증한다. 그 후, 다음 단계 S32에서, 글루 로직용 FPGA/에뮬레이션 유닛을 사용하여 온-칩 버스의 기능들 및 글루 로직을 포함하는 코어들 간의 상호 접속을 검증한다. 다음 단계 S33에서, 검증 방법은 코어-투-코어 통신용 시뮬레이션 테스트벤치들 및 SoC 레벨 임계 경로들을 사용하여 코어들의 타이밍을 검증한다. 최종 단계 S34에서, 전체 SoC 설계 검증이 전체 기능 시뮬레이션 테스트벤치들을 사용하고 어플리케이션 소프트웨어를 실행시킴으로써 이루어진다.

본 발명의 방법은 새로운 테스트 시스템의 전체 개념과 전자 설계 자동화(EDA) 환경과의 관계를 설명한 도 3에 도시된 장치로 구현된다. 도 3의 좌측 상부는 SoC(43)와 같은 반도체 장치들이 CAD 도구들을 사용하여 설계되는 EDA를 도시한 것이다. 도 3의 우측 하부에서, 본 발명의 설계 검증국(50)이 구현된다. 설계 검증국(50)은 SoC에 집적될 개별 코어들을 갖는 실제 반도체 IC들 뿐만 아니라 테스트 될 SoC(43)의 설계 환경에서 생성된 테스트 데이터 및 설계 데이터를 토대로 SoC 설계 검증을 실행한다.

이 실시예에서, SoC(43)는 기능 코어들(A, B, C)을 포함하는데, 자세한 구조는 도 4a에 도시되어 있다. EDA 환경(41)에서 SoC(43)를 설계한 후에, 설계 데이터 파일(45) 및 테스트 설계 파일(53)이 획득된다. 다양한 데이터 변환 공정들을 통해, 설계 데이터(45)는 설계된 반도체 집적 회로를 형성하는 각각의 게이트를 나타내는 물리 레벨 데이터로 변환된다. 물리 레벨 데이터를 토대로, 실제 SoC(49)가 반도체 IC 제조 공정(실리콘 공정)에서 생산된다. 그러나, 본 발명에서는 완전한 SoC를 직접 테스트하기 보다는 코어들 A, B 및 C와 같은 SoC (43)의 개별 코어들을 대표하는 분리 IC들이 설계 검증국(50)에서 사용된다.

SoC의 설계 단계를 통해 유도된 테스트 데이터(53)를 사용하여 테스트벤치 (54)에 의한 로직 시뮬레이션을 실행함으로써, 베릴로그/VCD 파일과 같은 데이터 파일(55)이 생성된다. 이는 개별 코어들 및/또는 SoC의 전체 시스템의 입출력 관계들을 보여 준다. 이하에 설명된 바와 같이, VCD 파일(55)의 데이터는 이벤트 기저 포맷으로 되어 있다. VCD 파일(55)의 데이터는 설계 검증국(50)에서 이벤트 파일들(58)로 전송된다. 설계 검증국(50)은 도 2의 상술된 절차에서의 테스트를 실행하기 위해 다수의 이벤트 테스터들(59) (도 4b의 검증 유닛들(66))을 포함한다.

설계 검증국(50)의 기본 구조의 일례가 소프트웨어/하드웨어 공동 개발/검증 (co-development/verification)을 위해 도 4b의 개략도에 도시되어 있다. 설계 검증국(50)은 테스트될 장치들의 핀에 따라 재구성될 수 있는 다수의 검증 유닛들 (VU) (66_1 - 66_N)을 포함한다. 검증 유닛들(66_1 - 66_N)은 평가될 SoC에 집적될 대응 코어들(A-N)의 기능과 회로 구조를 갖는 실리콘 IC들(68_1 - 68_N)에게 할당된다.

메인 시스템 CPU(62)는 검증 절차의 전체 동작을 제어한다. 메인 시스템 CPU(62) 및 검증 유닛들(66_1 - 66_N)은 시스템 버스(64)를 통해 접속된다. 검증 절차를 시작하기 전에, 메인 시스템 CPU(62)가 각각의 코어들(A-N)의 설계 단계로부터 유도된 설계 데이터(61) 및 테스트벤치 데이터(63)와 함께 제공된다.

설계 검증국(50)은 도 5에 더 상세히 도시되어 있는데, 여기서 설계 검증국은 기술된 목적들만을 위한 다수의 설계 검증국들(DVS_1 - DVS_6)로 도시되어 있다. 이 일례는 코어들(A-E) 및 글루 로직을 갖는 SoC가 설계 타당성으로 평가되는 케이스를 도시한 것이다. 이러한 실시예에서, 검증 기지국(DVS_1)은 " 버스 마스터 코어" (코어 A)를 테스트하도록 구성되고, 검증 기지국(DVS_2)은 " 프로세서 코어" (코어 B)를 테스트하도록 구성되고, 검증 기지국들(DVS_3, DVS_4)은 " 기능 특정 코어들" (코어들 C 및 D)을 테스트하도록 구성되고, 검증 기지국(DVS_5)은 " 메모리 코어" (코어 E)를 테스트하도록 구성된다. 유사하게, 검증 기지국(DVS_6)은 SoC의 " 글루 로직" 을 테스트하도록 구성된다. 본 발명에서, 상술된 코어들(A-E)은 설계 검증을 목적으로 분리 실리콘 IC들(68_1 - 68_5)로 공식화된다.

도 4a 및 도 5에 도시된 바와 같이, 시스템은 버스 기저 구조(bus based architecture)를 포함한다. 시스템 버스(64)는 데이터가 메인 시스템 CPU(62)로부터 검증 유닛들(66_1 - 66_5)로 전송될 수 있게 하는 VME, VXI, 또는 PCI 버스와 같은 산업 표준 버스(industry standard bus)일 수 있다. 시스템 핀들은 사용자에 의해 구성될 수 있다. 즉, 사용자는 개별 코어들(A-E)의 실리콘 IC들(68_1 - 68_5)의 입출력에 따라 검증 유닛(VU)들의 테스트 핀들을 그룹화할 수 있다. 실리콘 IC들(68_1 - 68_5)은 핀 일렉트로닉스 및 장치 로드 보드들(69_1 - 69_5) (이하 " 핀 일렉트로닉스" 라고 함)에 장착되고 상호 접속 버스(71)를 통해 서로 접속된다.

도 5에 도시된 바와 같이, 각각의 핀-그룹(할당된 검증 유닛)은 개별 코어들 및 SoC 상태의 모니터링 뿐만 아니라 데이터 흐름, 실리콘 코어들(68_1 - 68_5)에 대한 시뮬레이션 데이터의 어플리케이션, 응답 비교, 개별 블록들/코어들에 대한 다양한 작업들의 스케줄링(scheduling)을 제어하는 제어 CPU(67)를 포함한다. 제어 CPU들(67_1 - 67_6)은 서로 접속되어 있다. 제어 CPU들(67_1 - 67_6)은 또한 버스 시스템(64)을 통해 메인 시스템 CPU에 접속된다. 글루 로직을 위한 설계 검증국(DVS_6)에서, 동기화 유닛(75) 및 중재 유닛(arbitration unit) (76)이 메인 시스템 CPU(62)와 설계 검증국들(DVS_1 - DVS_6)의 제어 CPU들(67_1 - 67_6) 간의 데이터 전송을 향상시키기 위해 제공된다.

검증 공정에 앞서, 메인 시스템 CPU(62)는 개별 테스트벤치 데이터(78)를 초기화하고 테스트벤치 데이터를 검증 유닛들(66_1 - 66_N)에게 분배한다. 메인 시스템 CPU(62)는 사용자 인터페이스, 설계의 공동 검증을 위해 코어들에서의 소프트웨어 어플리케이션 실행, 및 검증 유닛들의 다중 분배 제어를 포함하는 설계 검증의 전체 절차를 제어한다. 각각의 설계 검증국(DVS)에서, 검증 유닛(VU) (66)은 코어의 대응 실리콘 IC(68)로의 테스트벤치 데이터를 토대로 생성된 테스트 패턴을 사용한다. 양호하게, 각각의 검증 유닛(VU) (66)은 이벤트 테스터로서 구성되는데 이에 대한 설명은 후에 기술될 것이다.

이벤트들(테스트 패턴들)은 도 5의 핀 일렉트로닉스들(69)을 통해 DUT에 적용된다. 핀 일렉트로닉스들(69)은 실리콘 IC(68; DUT)의 할당된 장치 핀들과 테스트 핀들을 물리적으로 접속시킨다. 기본적으로, 핀 일렉트로닉스들(69)은 검증 유닛(VU; 66)과 테스트될 실리콘 IC(68) 간의 인터페이스를 위해 인터페이스 회로를 포함한다. 예를 들어, 각각의 인터페이스 회로는 성능 보드 뿐만 아니라 하나 이상의 구동기와 비교기로 구성된다. 구동기는 DUT의 입력 핀에 테스트 패턴을 적용하고 비교기(도시되지 않음)는 DUT의 응답 출력을 예측 값과 비교한다. 성능 보드는 테스트 중에 DUT를 기계적으로 접속하는데 사용된다.

개별 코어들의 검증(Verification of Individual Cores)

본 발명에서, 개별 코어들(A-E)의 검증을 위해 개별 코어들의 실리콘 IC들 (68_1 - 68_5)이 사용된다. 이 실리콘 IC들(68)은 제조 파트너 회사들 뿐만 아니라 코어 제공자 회사들에 의해서도 일반적으로 사용된다. 전체 시스템은 도 4a와 도 5에 도시된 코어당 하나의 검증 유닛(66)을 매핑하도록 재구성된다. 검증을 위해, 개별 코어들의 테스트벤치들이 코어 각각의 입출력 정보와 함께 메인 시스템 CPU(62)에 로드된다.

메인 시스템 CPU(62)는 코어 당 하나의 검증 유닛(VU; 66)으로 시스템 핀들을 재구성하고 제어 CPU(67)를 할당한다. 시스템 성능을 강화하기 위해, 검증 유닛(VU; 66) 당 하나의 제어 CPU(67) 대신 핀 당 하나의 제어 CPU(67) 방식으로 상기 개념을 구현할 수 있음을 주목해야만 한다. 이러한 구현은 도 4a와 도 5에 도시된 시스템의 직접적이고 자연적인 강화(enhancement)인데 이에 대한 설명은 생략된다.

코어의 입출력들을 토대로, 구성 검증 유닛(66)은 2^N형태로 64 내지 256개의 핀들 사이 어디에서나 존재할 수 있다. 기본적으로 상기 핀들은 이벤트 테스터 채널들이고 구동/비교 동작을 허용한다. 도 4b와 도 5에 도시된 시스템은 이러한 핀들의 재구성을 허용하고 핀들을 개별 코어들위에 매핑한다. 따라서, 본질적으로, 전체 시스템은 도 5에 도시된 바와 같이 하나의 IP 또는 코어로 각각 매핑되는 다수의 검증 유닛들(66)로 구성된다. 따라서, 각각의 개별 코어에 대해, 할당된 VU(66)가 전용(dedicated) 이벤트 기저 검증 시스템이라고 여겨진다. 이벤트 기저 검증 시스템의 일례는 미국 특허 출원 제09/406,300호, " 이벤트 기저 반도체 테스트 시스템(Event Based Semiconductor Test System)" 에 기술되어 있고, 이후에 간단하게 설명될 것이다. 응답이 관측되어 검증 유닛(66)에 의해 시뮬레이션 데이터와 비교될 수 있을 뿐만 아니라, (기능 및 구조 검사를 위한) 코어의 이벤트 기저 시뮬레이션 벡터가 코어(실리콘 IC; 68)에 적용될 수도 있다.

코어 기능 및 타이밍 검증에 있어서, 메인 시스템 CPU(62)는 코어 시뮬레이션 테스트벤치 데이터를 관련 검증 유닛(VU; 66)의 제어 CPU(67)에 전달한다. 상기 데이터는 코어의 설계-시뮬레이션 테스트벤치이다. 상기 데이터는 신호 값이 0으로부터 1로 또는 1로부터 0으로 변경될 때의 순간들, 즉, 이벤트 기저 테스트 패턴들을 식별하기 위해 신호 값들과 타이밍 정보를 포함한다. 따라서, 변환이 필요 없고 데이터가 직접 코어 실리콘(68)에 적용된다.

데이터가 설계 시뮬레이션 데이터이면, 결함 없는 코어가 시뮬레이션에 의해 예측된 대로 정확하게 실행된다. 이러한 응답은 관측되어서 검증 유닛(66)의 제어 CPU(67)에 의해 비교된다. 시뮬레이션으로부터의 임의의 일탈(deviation)은 제어 CPU(67)에 의해 식별된다. 이는 검증 유닛(VU; 66)에 코어 IC의 임의의 오류가 존재함을 알려 준다. 이 단계는 SoC 레벨 설계 검증 전에 검증 유닛(VU; 66)에 대한 코어의 오류 없는 실리콘 IC를 갖게 한다.

또한 본 발명의 방법 및 장치가 사용자로 하여금 현재 시스템들 보다 훨씬 더 쉽게 코어의 오류를 디버그할 수 있게 함을 주목해야만 한다. 이는 본 발명의 검증 시스템의 환경이 고유 설계 시뮬레이션 환경과 동일하기 때문인데, 즉 이벤트 기저 데이터를 사용하는 EDA 환경이기 때문이다.

검증 유닛(Verification Unit; 이벤트 테스터)

상술된 바와 같이, 본 발명에 있어서 각각의 검증 유닛(66)은 이벤트 테스터로서 구성된다. 이벤트 테스터의 일례가 도 6을 참조하여 간단하게 설명된다. 보다 상세한 설명은 본 발명과 동일한 출원인이 소유한 상술된 미국 특허 출원 제09/406,300호, " 이벤트 기저 반도체 테스트 시스템" 에 기술되어 있다. 이벤트 테스터에서, 테스트 패턴은 기준 포인트로부터의 시간 길이 파라미터로서 0 에서 1로 또는 1에서 0으로의 신호 값 변경에 의해 기술된다. 전형적인 사이클 기저 테스트 데이터에 있어서, 테스트 패턴은 각각의 테스트 사이클에 대한 타이밍 데이터, 파형 데이터 및 벡터 데이터의 결합에 의해 묘사된다. 오늘날의 설계 자동화 도구들은 설계된 반도체 장치를 위한 로직 시뮬레이션을 실행할 때 이벤트 기저 테스트 데이터를 생성하기 때문에, 이벤트 기저 테스트 시스템은 반도체 장치의 설계 단계에서 생성된 시뮬레이션 데이터를 직접 사용할 수 있다.

도 6의 실시예에 있어서, 검증 유닛(66)은 핀-유닛 버스(63), 내부 버스 (85), 어드레스 시퀀서(88), 고장 메모리(87), 이벤트 메모리(90), 압축 해제 유닛 (92), 타이밍 카운트 및 스케일링 로직(93), 및 이벤트 생성 유닛(94)에 접속된 검증 유닛 기록 디코더(83)와 제어 CPU(67)를 포함한다. 검증 유닛(66)은 핀 일렉트로닉스들을 통해 코어를 포함하는 실리콘 IC(68)에 테스트 패턴을 제공한다.

검증 유닛 기록 디코더(83)는 메인 시스템 CPU(62)가 시스템 버스(64)에 그룹 선택 어드레스를 송신함으로써 검증 유닛들(66)을 구성할 수 있도록 검증 유닛 (66)의 식별명(identification)을 보여주기 위한 것이다. 내부 버스(85)는 하드웨어 이벤트 테스터의 버스이며, 어드레스 시퀀서(88), 고장 메모리(87), 압축 해제 유닛(92), 타이밍 카운트 및 스케일링 로직(93), 및 이벤트 생성 유닛(94)과 같은 대부분의 기능 블록들과 공통으로 접속된다.

상술된 바와 같이, 제어 CPU(67)는 메인 시스템 CPU(62)로부터의 코어 테스트벤치 데이터를 토대로 검증 유닛(66)의 다른 기능 블록들에게 명령들을 제공한다. 고장 메모리(87)는 어드레스 시퀀서(88)에 의해 제공된 어드레스 정보와 함께, 비교기(도시되지 않음)로부터의 코어의 실리콘 IC(68)의 고장 정보와 같은 테스트 결과들을 기억한다. 고장 메모리(87)에 기억된 정보는 코어들과 SoC의 고장 분석 단계에서 사용된다.

어드레스 시퀀서(88)는 이벤트 메모리(90)에게 어드레스 데이터를 제공한다. 이벤트 메모리(90)는 각각의 이벤트에 대한 타이밍 데이터를 기억한다. 예를 들어, 이벤트 메모리(90)는 2개의 분리된 방식들로 이벤트 데이터를 기억한다. 하나는 마스터 (참조) 클럭의 한 사이클의 정수 배인 타이밍 데이터를 기억하기 위한 것이고, 다른 하나는 기준 클럭의 한 사이클의 소수(fraction) 또는 소수들인 타이밍 데이터를 기억하기 위한 것이다.

바람직하게는, 이벤트 메모리(90)의 타이밍 데이터가 필요한 메모리 용량을 감소시키기 위해 압축된다. 압축 해제 유닛(92)은 이벤트 메모리(90)로부터 압축된 데이터를 수신하여서 압축 해제 공정을 통해 타이밍 데이터를 재생성한다. 압축 해제 타이밍 데이터는 타이밍 카운트 및 스케일링 로직(93)에 제공된다.

타이밍 카운트 및 스케일링 로직(93)은 이벤트 메모리(90)로부터의 타이밍 데이터를 근거로 현재의 이벤트를 생성한다. 전체 타이밍 데이터는 현재의 타이밍 데이터를 이전 타이밍 데이터와 합산함으로써 생성될 수 있다. 타이밍 카운트 및 스케일링 로직(93)은 또한 스케일링 팩터에 비례하여 타이밍 데이터를 수정하도록 동작한다. 타이밍 데이터의 스케일링 동작은 타이밍 데이터(각각의 델타 시간 또는 절대 시간)에 스케일링 팩터를 곱함으로써 이루어진다. 이벤트 생성 유닛 (94)은 타이밍 카운트 및 스케일링 로직(93)으로부터의 전체 타이밍 데이터를 근거로 실 시간 이벤트 신호들을 생성한다. 이벤트 생성 유닛(94)은 핀 일렉트로닉스들(69)에게 이벤트 신호들(테스트 패턴들)을 제공한다.

인터페이스, 온-칩 버스 및 글루 로직 검증(Interface, On-Chip Bus and Glue Logic Verification)

SoC 설계의 대부분은 미리 설계된 코어들로 구성되지만, 다양한 코어들을 접속하도록 할 뿐만 아니라 소정의 특정 기

능들을 실행하도록 코어 인테그레이터(SoC 설계자)에 의해 설계된 항상 일정한 로직이 존재한다. 상기 로직은 일반적으로 " 글루 로직" 이라고 한다. 전형적으로, 글루 로직은 주문 설계를 통해 구현되어져 왔지만, 최근에는 내장된 FPGA(field programmable gate array)이 이러한 로직을 구현하도록 제안되어 왔다. 상술된 바와 같이, 현재의 기술에 있어서 매우 불완전한 검증은 상기 로직으로 끝났다.

제안된 방법에 있어서, 상기 로직의 검증은 글루 로직 검증을 위한 설계 검증국(DVS $_6$)으로 도 5에 도시된 전용 서브-시스템에 의해 이루어 졌다. 기본적인 방법은 다음과 같다:

(1) SoC 온-칩 버스를 모델링하기 위해 도 5에 도시된 바와 같이 다양한 실리콘 IC들(68)을 접속하는 상호 접속 버스(71)를 사용. 상호 접속 버스(71)는 온-칩 버스의 행동을 모델링하는 다양한 코어들(A-E)을 접속하는 시스템 버스이다. 이 버스는 SoC 레벨(하나의 코어에서부터 다른 코어로)에서의 명령 및 데이터 흐름을 설계 검증국 레벨(하나의 검증 유닛으로부터 다른 검증 유닛으로)에서의 명령 및 데이터 흐름으로 매핑한다. 따라서, 이 버스는 개별 코어들의 각각의 인터페이스의 모든 데이터 트랜잭션들 뿐만 아니라 SoC 온-칩 버스의 임의의 요구/승인 프로토콜을 포착한다.

(2) 전용 서브-시스템에 글루 로직을 구현하는 FPGA(field programmable gate array)를 사용. 다른 기법(alternate approach)은 전용 서브-시스템에 글루 로직을 에뮬레이트하는 것이다. 2가지 기법들 모두가 도 7과 도 8에 각각 도시되어 있다.

도 7은 에뮬레이터 서브-시스템을 도시한 것이다. 이 기법에서, 임의의 상업용 에뮬레이터 시스템이 사용될 수 있다. 도 7에서, 에뮬레이터(72)는 글루 로직의 합성 가능 RTL과 글루 로직 테스트벤치 파일(77)의 테스트벤치 데이터와 함께 로드된다. 합성 유닛 및 중재 유닛들은 다른 검증 유닛들(66)과의 인터페이스를 위해 상업용 에뮬레이터와 함께 사용된다. 제어 CPU(67)는 메인 시스템 CPU(62)와의 동기화 및 통신 작업들을 실행한다.

도 8은 FPGA 접근을 도시한 것이다. 이러한 접근에 있어서, 글루 로직 설계는 FPGA(73)를 사용하여 구현되고 실리콘 IP 또는 코어로서 FPGA(73)를 처리한다. 글루 로직이, 내장된 FPGA에 의해 SoC에서 구현되면, FPGA(73)는 내장된 FPGA(글루 로직)의 스탠드 얼론(stand-alone) 복사품이다. FPGA(73)는 독립 IP로서 사용되고 전용 검증 유닛(VU)에 할당된다.

글루 로직이 주문 설계에 의해 SoC에서 구현되면, 글루 로직의 RTL은 전용 검증 유닛에서 사용되는 스탠드 얼론 FPGA에서 구현된다. 이 경우에, 대부분의 시간에 FPGA의 속도가 SoC의 주문형 글루 로직보다 더 느리다. 따라서, 전용(dedicated) 유닛은 추가의 동기화 유닛(75)과 버스 중재 유닛(76)을 필요로 한다. 느린 동작 속도 외에도 검증 유닛은 동작이 임의의 다른 검증 유닛과 유사할 뿐만 아니라 임의의 다른 검증 유닛과 동일하다.

타이밍 검증(Timing Verification)

개별 코어들의 기능, 인터페이스 및 글루 로직이 검증되면, 타이밍 검증이 SoC 레벨 임계 경로들에서 검사된다. 도 2의 단계 31 및 32의 완료 후에, SoC의 모든 개별 부분들과 상호 접속이 본 발명의 설계 검증국에서 사용됨을 주목해야만 한다. 개별 코어들의 타이밍이 검증되었을 뿐만 아니라 개별 코어들의 기능 및 글루 로직이 검증되었다. 따라서, 전체 어플리케이션들 뿐만 아니라 SoC 레벨 시뮬레이션 테스트벤치가 전체 시스템에서 실행될 수 있고, 임의의 에러의 경우, 코어를 집적할 때 에러가 나타난다는 것을 알 수 있다.

본 발명의 방법에 있어서, 코어-투-코어 타이밍의 정확성과 SoC 레벨에서의 타이밍 임계 경로들의 정확성을 검증하는 적은 수의 SoC 레벨 시뮬레이션 벡터들을 실행하는 것이 바람직하다. 상기 목적을 위해, SoC 레벨 시뮬레이션 테스

트벤치가 메인 CPU에 로드된다. SoC 설계 동안, 상기 시뮬레이션 테스트벤치들이 개발되어 설계의 타이밍 임계 경로들을 실행시킨다. 이러한 테스트벤치들(벡터들)의 데이터는 이벤트 형태이고, 오늘날의 기술에 있어서 베릴로그/VHDL 시뮬레이터로부터 획득된 VCD(value change dump) 포맷이 일반적으로 사용된다.

테스트벤치 데이터의 벡터들은 SoC의 상이한 부분들을 접속하는 SoC의 다양한 타이밍 임계 경로들을 실행시킨다. 상술된 바와 같이, 본 발명의 설계 검증국은 SoC의 모든 구성 요소들을 갖는데, 타이밍 검증을 위한 시뮬레이션 테스트벤치가 실행되어 시뮬레이션과 동일한 결과들을 생성할 것으로 예측된다. 시뮬레이션 결과들로부터의 임의의 변형은 에러를 식별하는데, 에러는 설계 시뮬레이션 환경과 동일한 본 발명의 이벤트 기저 환경에서 쉽게 디버그된다.

SoC 또는 전체 설계 확인의 검증(Verification of SoC or Overall Design Validation)

시스템으로서의 SoC 전체 기능 검증을 위해, 설계 시뮬레이션 중에 개발된 SoC 레벨 기능 벡터들이 설계 검증국에서 실행된다. 또한 상기 벡터들은 이벤트 형태이다. SoC 설계(베릴로그/VHDL RTL 모델 또는 행동 모델)에서 실행되는 소프트웨어 어플리케이션에 의해 벡터들이 수회 생성된다. 그러나, 벡터들은 SoC의 상이한 경로들을 동시에 또는 상이한 시간에 실행시키므로, SoC 전체 행동은 결합된 응답에 의해 결정된다.

어플리케이션 프로그램이 C/C++ 언어와 같은 하이 레벨 언어로 되어 있거나 이진 형태로 되어 있을 때, API(application program interface) 및 PLI(program language interface)가 도 9에 도시된 바와 같이 외부 세계와 메인 시스템 CPU(62) 간의 통신을 위해서 뿐만 아니라 메인 시스템 CPU(62)에 상기 프로그램들을 로드하기 위해 필요하다.

이를 달성하기 위해, 메인 시스템 CPU(62)는 (도 5 및 도 9에 멀티-BP로 도시된) 다중 버스 프로토콜을 갖는 다중 분배 제어(multiple distribution control)를 포함한다. 메인 시스템 CPU(62)는 어플리케이션 작업을 다수의 서브-작업들로 분할하고 스케줄링해서(schedule), 개별 코어들로 매핑되는 상이한 검증 유닛들(66)에게 서브-작업들을 할당하도록 어플리케이션 작업(소프트웨어 어플리케이션)에 대한 " 포크(Fork)" 동작을 실행한다. 상기 " 포크" 동작은 베릴로그/VHDL 또는 심지어 C/C++ 언어와 같은 하이 레벨 언어로 된 어플리케이션 소프트웨어에 대하여서 실행됨을 주목해야만 한다. 따라서, 다중 분배 제어를 갖는 시스템 컴파일러가 다수의 검증 유닛들(66)로 이루어진 분배 계산 환경에서 실행시키기 위해 어플리케이션 작업에 대하여 " 포크" 동작을 실행시킬 수 있다.

이러한 " 포크" 동작 후에, 서브-작업들은 시스템 버스(64)를 통해 개별 검증 유닛들(66)에게 분배된다. 제어 CPU(67), 중재 유닛(76) 및 동기화 클럭 유닛 (75)은 메인 시스템 CPU(62)로부터 개별 검증 유닛들(66)의 제어 CPU(67)로의 통신 및 에러 없는 데이터 전송을 허용한다. 제어 CPU(67), 중재 유닛(76) 및 동기화 클럭 유닛(75)을 갖는 구조는 도 9에 도시되어 있다.

서브-작업 할당을 근거로, 제어 CPU들(67)은 이벤트 기저 벡터들을 개별 코어들에게 적용시키고 그로부터의 응답을 수집한다. 이 응답은 에러 없는 데이터 전송을 위해 버스 제어 CPU, 버스 중재 유닛 및 동기화 유닛을 다시 사용하여 메인 시스템 CPU(62)에게 전달된다. 메인 시스템 CPU(62)는 " 연결(Join)" 동작을 실행하여 다양한 응답들을 병합하고 SoC 레벨 응답을 형성한다. 상기 응답은 SoC가 정확한 동작을 실행했는지를 결정하기 위해 시뮬레이션 응답과 비교된다. 이것이 어플리케이션 실행이면, 그 때 상기 응답은 어플리케이션의 예측된 결과이다. 예를 들어, 비디오 어플리케이션 실행이 픽쳐 프레임의 디스플레이를 야기하게 된다. 환경이 이벤트 기저 고유 설계 환경과 동일한 것이기 때문에 시뮬레이션 데이터 또는 예측된 어플리케이션 출력의 변경은 제어 CPU(67)에 의해 식별되고 설계 엔지니어에 의해 쉽게 디버그된다.

고정 또는 성능 보드(Fixture or Performance Board)

본 발명의 설계 검증국은 실리콘 코어들(68)과 글루 로직(FPGA)이 인터페이스하는 성능 보드를 필요로 한다. 도 5 내지 도 9의 실시예에서, 장치 로드 보드 또는 성능 보드(69)가 코어(각각의 설계 검증국(DVS)) 마다 제공된다. 도 10의 블록도는 성능 보드에 대한 다른 일례의 구조를 도시한 것이다. 이러한 실시예에서, 성능 보드(90)는 테스트될 모든 코어들 및 글루 로직을 포함한다. 커넥터들(95)은 검증 유닛(66)과 실리콘 코어(68)를 연결하기 위해 그 사이에 제공된다.

성능 보드(90)는 다층 프린트 회로 보드(multi-layered printed circuit board)인 종래의 테스터의 성능 보드와 매우 유사하다. 상기 성능 보드(90)와 테스터 성능 보드와의 주요한 차이는 테스터 성능 보드가 오직 하나의 DUT만을 포함하지만, 본 발명의 설계 검증국의 상기 성능 보드(90)는 모든 코어들의 실리콘 IC들(68)과 글루 로직(FPGA)을 포함한다는 점이다.

도 11은 다수의 검증 유닛들(VU)이 메인 시스템 컴퓨터에 의해 직접 제어되는 본 발명의 설계 검증국의 다른 실시예를 도시한 것이다. 이러한 실시예에서는 이전 실시예들과 달리, 각각의 설계 검증국이 제어 CPU를 포함하지 않고 시스템 버스(64)를 통해 메인 시스템 컴퓨터(62)에 의해 집적 제어된다. 따라서, 동기화, 코어들의 응답 평가, 타이밍 평가, 및 전체 SoC 평가 등과 같은 모든 작업들은 메인 시스템 컴퓨터(62)에 의해 실행된다.

양호한 실시예가 특정하게 도시되고 기술되었지만, 상술된 설명의 견지에서 또한 본 발명의 정신과 목적을 벗어나지 않고 첨부된 청구항들의 범위 내에서 본 발명의 다양한 변형 및 변화가 가능함을 알 수 있다.

발명의 효과

본 발명은 각각의 코어 기능, 코어들 간의 상호 접속, 및 전체 시스템 성능에 대한 SoC 설계 검증을 실행할 수 있는 SoC 설계 검증 방법 및 장치를 제공한다.

또한, 본 발명은 빠른 속도와 저렴한 비용으로 SoC 설계 검증을 실행할 수 있는 SoC 설계 검증 방법 및 장치를 제공한다.

또한, 본 발명은 SoC의 전체 기능을 검증하기 위해 SoC 설계 검증이 행해지는 설계 검증국을 제공한다. 본 발명은 내장 코어 기저 시스템-온-칩의 설계 검증의 현재의 어려움들을 해결하는 새로운 설계 검증 또는 전체 기능 검증을 위한 방법 및 장치를 제공한다. 본 명세서에 설명된 시스템 구조는 매우 효율적이고 비용이 적게 들며 임의의 이전 시스템과 기본적으로 상이하다.

본 발명의 검증 시스템의 환경이 고유 설계 시뮬레이션 환경과 동일한 이벤트 기저 데이터를 사용하는 EDA 환경이기 때문에 본 발명은 사용자로 하여금 현재 시스템들 보다 훨씬 더 쉽게 SoC의 코어들의 오류를 디버그할 수 있게 한다.

(57) 청구의 범위

청구항 1.

다수의 기능 코어들이 집적되어 있는 내장 코어 기저 SoC(system-on-a-chip)의 설계 검증(design validation)를 위한 방법에 있어서,

코어 제공자(core provider)에 의해 제공된 코어 각각의 실리콘 IC 및 시뮬레이션 테스트벤치들(simulation testbenches)을 사용하여, SoC에 집적될 개별 코어들을 검증하는 단계;

SoC 설계자에 의해 개발된 시뮬레이션 테스트벤치들과 글루 로직(glue logic)의 FPGA/에뮬레이션(emulation)을 사용하여 상기 개별 코어들 간의 인터페이스, 상기 코어들의 온-칩 버스들(on-chip buses), 및 글루 로직을 검증하는 단계;

코어-투-코어 타이밍(core-to-core timings)과 SoC 레벨 타이밍 임계 경로들을 검증하는 단계; 및

전체 SoC 및 어플리케이션 실행(application runs)의 시뮬레이션 테스트벤치를 사용하여 전체 설계 검증을 실행하는 단계

를 포함하는 설계 검증 방법.

청구항 2.

제1항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지는 설계 검증 방법.

청구항 3.

제1항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지되, 상기 검증 유닛들의 테스트 핀들이 테스트될 상기 실리콘 IC들의 입/출력 핀들과 관련하여 구성되는 설계 검증 방법.

청구항 4.

제1항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지되, 각각의 상기 검증 유닛들은 값들의 변경 및 타이밍에 의해 테스트 패턴을 기술하는 이벤트 데이터를 토대로 테스트 패턴을 생성하는 이벤트 기저 테스터로서 구성되는 설계 검증 방법.

청구항 5.

제4항에 있어서,

상기 개별 코어들의 상기 시뮬레이션 테스트 벤치가 이벤트 기저 데이터 포맷을 가지며, 이로써 이벤트 기저 테스터인 상기 검증 유닛에 의해 상기 SoC의 코어에 있어서의 오류 디버그를 용이하게 하는 설계 검증 방법.

청구항 6.

다수의 기능 코어들이 집적되어 있는 내장 코어 기저 SoC(system-on-chip)의 설계 검증을 위한 방법에 있어서,

SoC로 집적될 대응 코어의 회로 구조를 각각 갖는 다수의 실리콘 IC들을 생성하는 단계;

다수의 검증 유닛들을 제공하고 각각의 상기 검증 유닛을 상기 코어의 각각의 상기 실리콘 IC에 할당하는 단계;

상기 코어들 간의 접속을 위해 상기 SoC에서 설계된 온-칩 버스(on-chip bus)를 모델링하는 상호 접속 버스로 상기 코어들을 상호 접속하는 단계; 및

테스트 패턴들을 상기 실리콘 IC들에게 적용시키고 상기 실리콘 IC들의 응답 출력을 모니터함으로써 상기 SoC에 집적될 상기 코어들을 검증하는 단계

를 포함하되,

상기 데이트 패턴들은 코어 제공자에 의해 제공된 시뮬레이션 테스트벤치 데이터를 사용함으로써 직접적으로 제공되는 설계 검증 방법.

청구항 7.

제6항에 있어서,

에뮬레이터에 의해 상기 인터페이스 및 글루 로직을 에뮬레이트함으로써 상기 SoC에 집적될 코어들 및 글루 로직 사이의 인터페이스를 검증하는 단계를 더 포함하는 설계 검증 방법.

청구항 8.

제6항에 있어서,

FPGA(field programmable gate array)로 상기 인터페이스 및 글루 로직의 기능들을 구현함으로써 상기 SoC에 집적될 코어들 및 글루 로직 간의 인터페이스를 검증하는 단계를 더 포함하는 설계 검증 방법.

청구항 9.

제6항에 있어서,

SoC 레벨 테스트벤치 데이터를 토대로 생성된 테스트 자극을 상기 개별 코어들의 실리콘 IC들에게 제공함으로써 코어-투-코어 타이밍과 SoC 레벨 타이밍 임계 경로들을 검증하는 단계를 더 포함하는 설계 검증 방법.

청구항 10.

제6항에 있어서,

전체 SoC의 시뮬레이션 테스트벤치들과 어플리케이션 실행들을 사용함으로써 전체 설계 검증을 실행하는 단계를 더 포함하는 설계 검증 방법.

청구항 11.

제6항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지는 설계 검증 방법.

청구항 12.

제6항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지되, 각각의 상기 검증 유닛들은 값들의 변경 및 타이밍에 의해 테스트 패턴을 기술하는 이벤트 기저 테스트 패턴을 생성하는 이벤트 기저 테스터로서 구성되는 설계 검증 방법.

청구항 13.

제12항에 있어서,

상기 개별 코어들의 상기 시뮬레이션 테스트벤치가 이벤트 기저 데이터 포맷을 가지며, 이로써 상기 이벤트 기저 테스터인 상기 검증 유닛에 의해 상기 SoC의 상기 코어들에 있어서의 오류 디버그가 용이하게 되는 설계 검증 방법.

청구항 14.

제6항에 있어서,

다수의 검증 유닛들을 상기 개별 코어들의 상기 실리콘 IC들에게 할당한 후에 상기 검증 단계들이 행해지되, 상기 검증 유닛들의 테스트 핀들은 테스트될 상기 실리콘 IC들의 입/출력 핀들과 관련하여 구성되는 설계 검증 방법.

청구항 15.

다수의 기능 코어들이 집적되어 있는 내장 코어 기저 SoC(system-on-chip)의 설계 검증을 위한 장치에 있어서,

사용자와 인터페이스하고 설계 검증을 위한 장치의 전체 동작을 제어하기 위한 메인 시스템 컴퓨터;

상기 메인 시스템 컴퓨터로부터 테스트벤치 데이터를 수신하고 SoC에 집적될 다수의 기능 코어들을 테스트하기 위해 상기 테스트벤치 데이터를 사용하여 테스트 패턴들을 생성하며, 상기 메인 시스템 컴퓨터로부터의 상기 테스트벤치 데이터를 수신하는 제어 컴퓨터를 각각 포함하는 다수의 검증 유닛들; 및

상기 다수의 검증 유닛들과 상기 메인 시스템 컴퓨터를 인터페이스하는 시스템 버스

를 포함하되,

다수의 실리콘 IC들은 상기 검증 유닛들로부터 상기 테스트 패턴을 수신하고 상기 검증 유닛들과 메인 시스템 컴퓨터에 의해 평가될 응답 출력들을 생성하기 위해 상기 검증 유닛들과 접속되며, 상기 실리콘 IC들은 상기 SoC에 집적될 상기 기능 코어들과 동일한 내부 구조를 갖는 설계 검증 장치.

청구항 16.

제15항에 있어서,

각각의 상기 검증 유닛의 상기 제어 컴퓨터가 상기 시스템 버스를 통해 상기 메인 시스템 컴퓨터로부터 상기 테스트벤치 데이터를 토대로 상기 검증 유닛에 할당된 상기 실리콘 IC를 위한 상기 테스트 패턴들을 생성하고 상기 실리콘 IC의 상기 응답 출력을 평가하는 설계 검증 장치.

청구항 17.

제15항에 있어서,

각각의 상기 검증 유닛이 상기 실리콘 IC들 중 하나에 할당되는 설계 검증 장치.

청구항 18.

제15항에 있어서,

각각의 상기 검증 유닛이 상기 실리콘 IC들 중 하나에 할당되며, 상기 검증 유닛들의 테스트 핀들은 테스트될 상기 실리콘 IC들의 입/출력 핀들과 관련하여 구성되는 설계 검증 장치.

청구항 19.

제15항에 있어서,

각각의 검증 유닛이 한 그룹의 테스트 핀들을 가지며, 상기 검증 유닛의 핀 구성은 테스트될 상기 실리콘 IC의 핀 구성에 따라 자유롭게 변형되는 설계 검증 장치.

청구항 20.

제15항에 있어서,

각각의 검증 유닛이 한 그룹의 테스트 핀들을 갖고 상기 검증 유닛의 핀 구성은 테스트될 상기 실리콘 IC의 핀 구성에 따라 자유롭게 변형되며, 상기 그룹의 사이즈가 상기 검증 유닛에 할당된 상기 실리콘 IC의 입/출력 핀들의 수를 토대로 상기 메인 시스템 CPU에 의해 결정되는 설계 검증 장치.

청구항 21.

제15항에 있어서,

각각의 상기 검증 유닛이 상기 시스템 버스를 통해 상기 메인 시스템 컴퓨터로부터 상기 테스트벤치 데이터를 수신하고 상기 검증 유닛에 할당된 상기 실리콘 IC에 대한 상기 테스트 패턴들을 생성하며 상기 실리콘 IC의 상기 응답 출력을 평가하는 제어 컴퓨터를 포함하되, 상기 제어 컴퓨터는 상기 검증 유닛의 각각의 테스트 핀을 위해 제공되는 설계 검증 장치.

청구항 22.

제15항에 있어서,

상기 검증 유닛이 상기 대응 기능 코어를 검증하기 위해 할당된 상기 실리콘 IC를 평가하고, 상기 검증 유닛들은 에뮬레이터로 상기 인터페이스 및 글루 로직을 에뮬레이트함으로써 상기 SoC에 집적될 상기 실리콘 IC들 및 글루 로직 간의 인터페이스를 더 평가하는 설계 검증 장치.

청구항 23.

제15항에 있어서,

상기 검증 유닛이 상기 대응 기능 코어를 검증하기 위해 할당된 상기 실리콘 IC를 평가하고, 상기 검증 유닛들은 FPGA로 인터페이스 및 글루 로직의 기능들을 구현함으로써 상기 SoC에 집적될 상기 실리콘 IC 및 글루 로직들 간의 인터페이스를 더 평가하는 설계 검증 장치.

청구항 24.

제15항에 있어서,

상기 메인 시스템 컴퓨터와 상기 검증 유닛들이 상기 개별 코어들을 대표하는 상기 실리콘 IC들에게 SoC 레벨 테스트벤치 데이터를 토대로 생성된 테스트 자극을 제공함으로써 코어-투-코어 타이밍과 SoC 레벨 타이밍 임계 경로들을 검증하는 설계 검증 장치.

청구항 25.

제15항에 있어서,

상기 메인 시스템 컴퓨터와 상기 검증 유닛들이 전체 SoC의 시뮬레이션 테스트벤치들과 어플리케이션 실행들을 사용함으로써 전체 설계 검증을 검증하는 설계 검증 장치.

청구항 26.

제15항에 있어서,

상기 메인 시스템 컴퓨터와 상기 검증 유닛들이 전체 SoC의 시뮬레이션 테스트벤치들과 어플리케이션 실행들을 사용함으로써 전체 설계 확인을 검증하며, 상기 메인 시스템 컴퓨터는 계산들을 다중 서브-작업들로 분할하고 상기 서브-작업들을 분배 방식으로 상기 다수의 검증 유닛들에게 할당함으로써 상기 검증 유닛들에게 분할 SoC 소프트웨어 어플리케이션 계산들을 분배하는 설계 검증 장치.

청구항 27.

제15항에 있어서,

상기 메인 시스템 컴퓨터와 상기 검증 유닛들이 전체 SoC의 시뮬레이션 테스트벤치들과 어플리케이션 실행들을 사용함으로써 전체 설계 확인을 검증하며, 상기 메인 시스템 컴퓨터는 계산들을 다수의 서브-작업들로 분할하고 상기 서브-작업들을 분배 방식으로 상기 다수의 검증 유닛들에게 할당함으로써 상기 검증 유닛들에게 분할 SoC 소프트웨어 어플리케이션 계산들을 분배하고, 상기 메인 컴퓨터가 임의의 에러/고장을 결정하기 위해 SoC 레벨 응답을 형성하도록 상기 검증 유닛들로부터의 상기 실리콘 IC들의 응답들을 결합하는 설계 검증 장치.

청구항 28.

제15항에 있어서,

각각의 상기 검증 유닛의 상기 대응 실리콘 IC를 보유하는 성능 보드를 더 포함하는 설계 검증 장치.

청구항 29.

제15항에 있어서,

테스트 하에 모든 실리콘 IC들과 글루 로직을 보유하는 성능 보드를 더 포함하는 설계 검증 장치.

청구항 30.

제15항에 있어서,

각각의 상기 검증 유닛들이 값들의 변경 및 타이밍에 의해 테스트 패턴을 기술하는 이벤트 데이터를 토대로 상기 테스트 패턴을 생성하는 이벤트 기저 테스터로서 구성되는 설계 검증 장치.

청구항 31.

제30항에 있어서,

상기 개별 코어들의 상기 시뮬레이션 테스트벤치가 이벤트 기저 데이터 포맷을 가지며, 이로써 상기 이벤트 기저 테스터인 상기 검증 유닛에 의해 상기 SoC의 상기 코어들의 오류 디버그가 용이하게 되는 설계 검증 장치.

청구항 32.

제15항에 있어서, 상기 검증 유닛 각각은,

현재의 이벤트와 선정된 기준 포인트 간의 시간 차이며, 기준 클럭 주기의 정수 배수(정수 부분 데이터)와 기준 클럭 주기의 소수(소수 부분 데이터)로 형성되는 각 이벤트의 타이밍 데이터를 기억하기 위한 이벤트 메모리;

상기 이벤트 메모리를 액세스하기 위해 어드레스 데이터를 생성하기 위한 어드레스 시퀀서;

상기 정수 부분 데이터로 곱해진 상기 기준 클럭 주기만큼 지연된 이벤트 개시 신호를 생성하기 위한 이벤트 카운트 로직;

상기 테스트 패턴을 공식화하기 위해 이벤트 카운트 로직 및 소수 부분 데이터로부터의 이벤트 개시 신호를 토대로 각각의 이벤트를 생성하기 위한 이벤트 생성 유닛; 및

상기 검증 유닛을 상기 실리콘 IC의 상기 핀들에게 할당하기 위해 검증 유닛의 어드레스를 검출하기 위한 검증 유닛 기록 디코더

를 포함하는 이벤트 기저 테스터로서 구성되는 설계 검증 장치.

청구항 33.

다수의 기능 코어들이 집적되어 있는 내장 기저 SoC(system-on-chip)의 설계 검증을 위한 장치에 있어서,

사용자와 인터페이스하고 설계 검증을 위한 장치의 전체 동작을 제어하기 위한 메인 시스템 컴퓨터;

상기 메인 시스템 컴퓨터로부터의 테스트벤치 데이터를 수신하고 SoC에 집적될 다수의 기능 코어들을 테스트하기 위해 상기 테스트벤치 데이터를 사용하여 테스트 패턴들을 생성하는 다수의 검증 유닛들; 및

상기 다수의 검증 유닛들과 상기 메인 시스템 컴퓨터를 인터페이스하는 시스템 버스

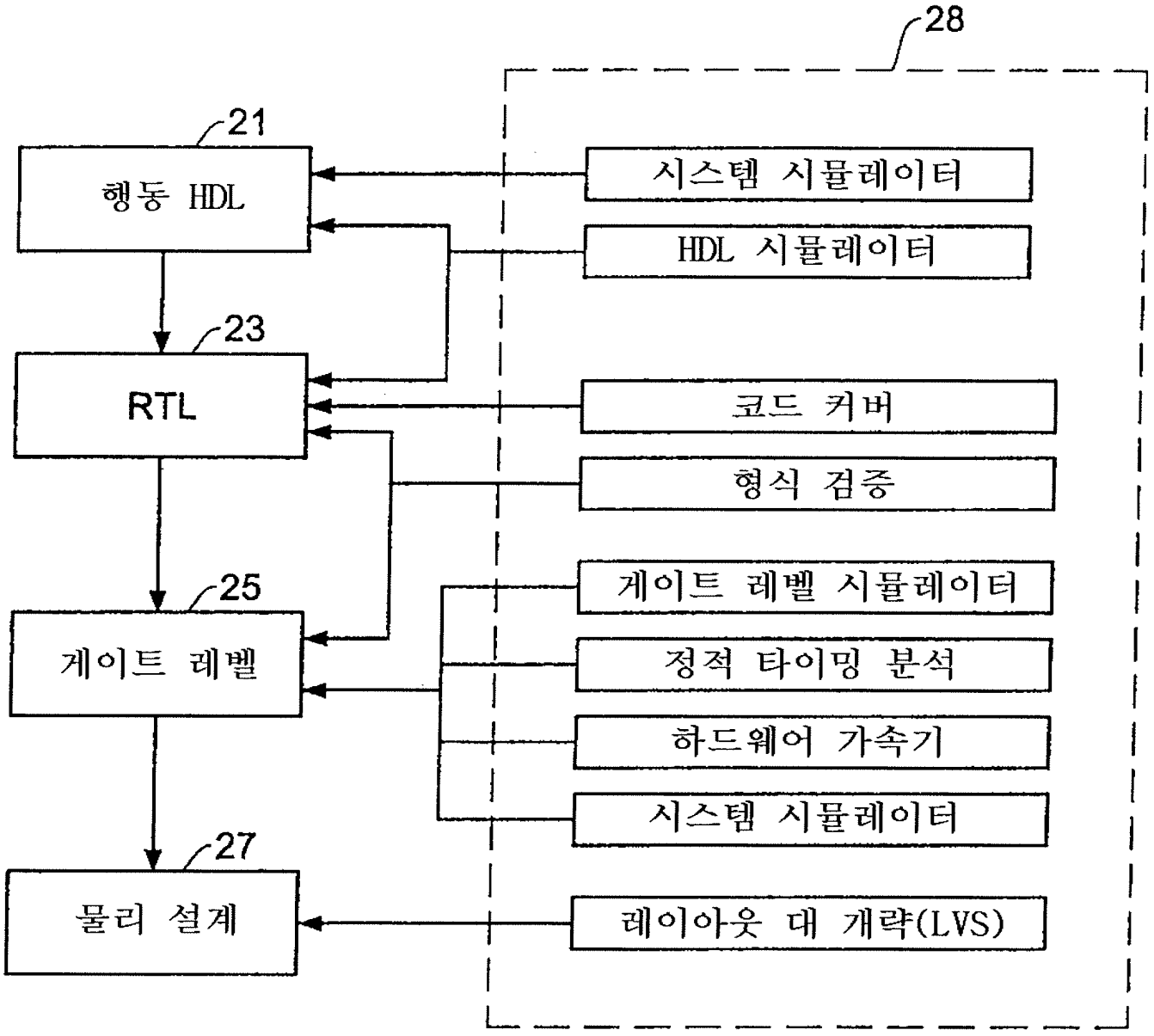
를 포함하되,

다수의 실리콘 IC들은 상기 검증 유닛들로부터 상기 테스트 패턴을 수신하기 위해 상기 검증 유닛들을 연결하여 상기 검증 유닛들 및 메인 시스템 컴퓨터에 의해 평가될 응답 출력들을 생성하고, 상기 실리콘 IC들이 상기 SoC에 집적될 상기 기능 코어들과 동일한 내부 구조를 가지며;

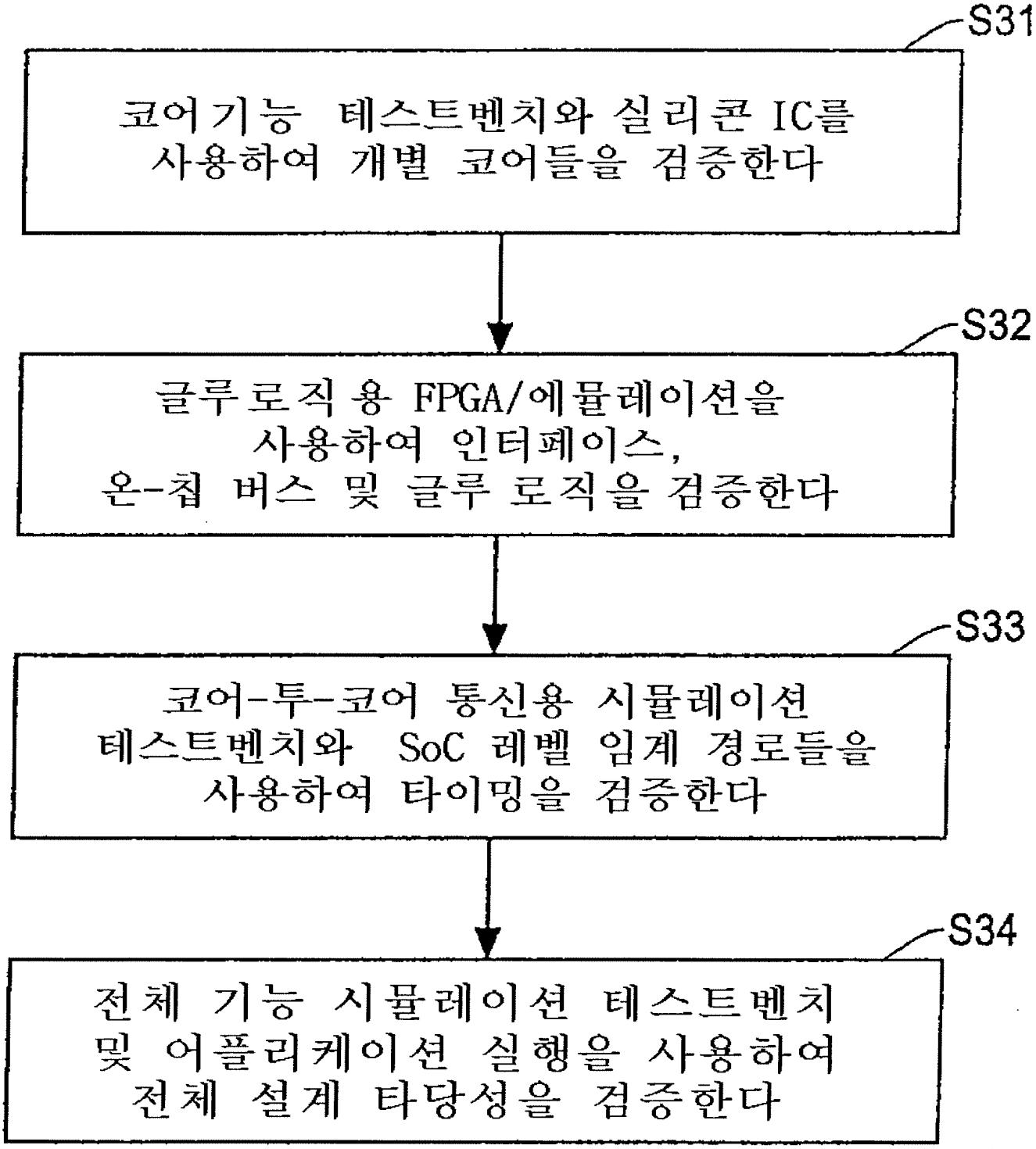
상기 메인 시스템 컴퓨터는 상기 실리콘 IC들에게 제공될 테스트 패턴들을 생성하고, 상기 실리콘 IC들의 응답 출력들을 평가하고, 상기 SoC의 타이밍 및 인터페이스 평가를 실행하고, 상기 SoC의 전체 설계 검증을 하는 모든 작업들을 실행시키는 설계 검증 장치.

도면

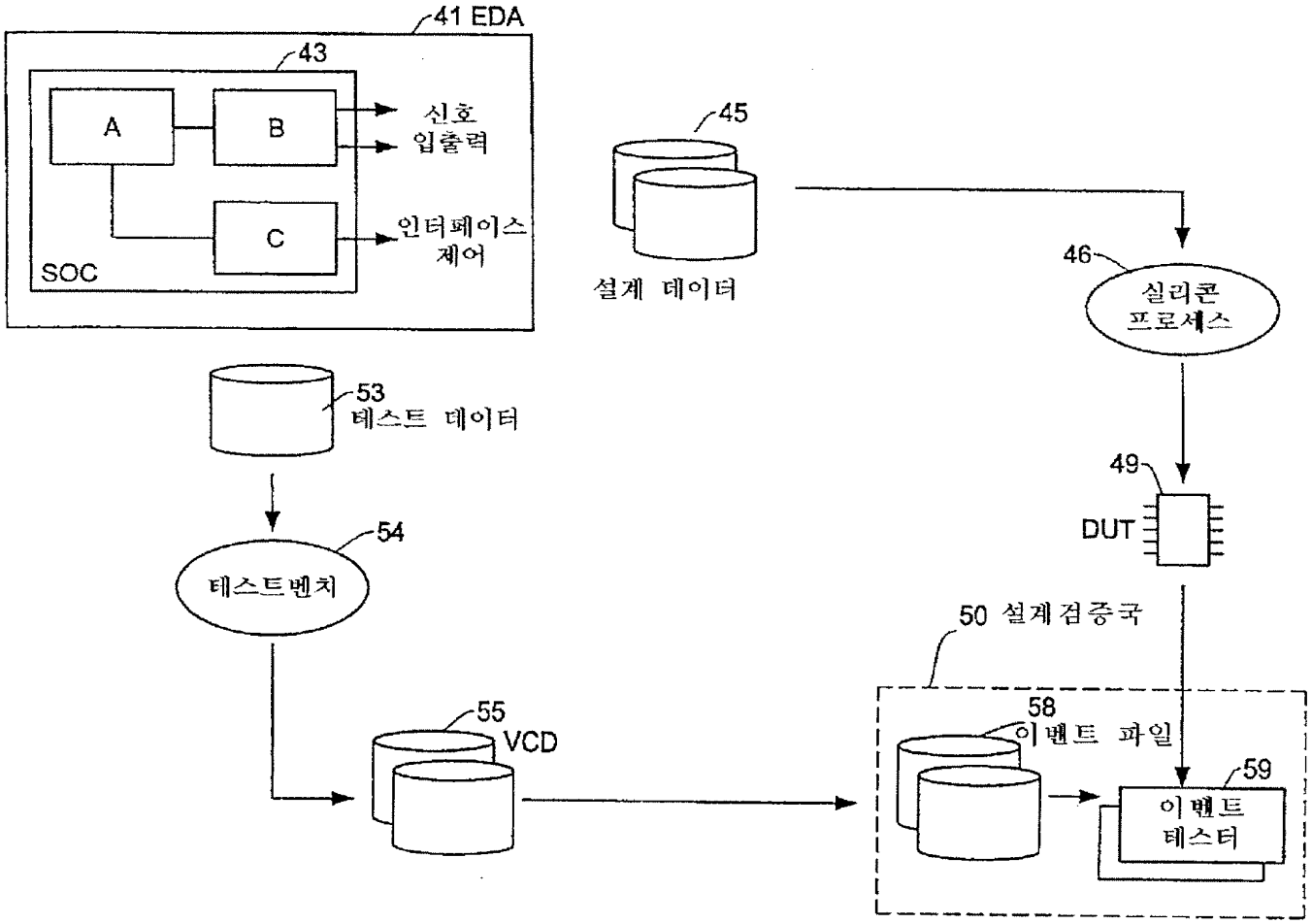
도면 1


28
21
행동 HDL
23
RTL
25
게이트 레벨
27
물리 설계
시스템 시뮬레이터
HDL 시뮬레이터
코드 커버
형식 검증
게이트 레벨 시뮬레이터
정적 타이밍 분석
하드웨어 가속기
시스템 시뮬레이터
레이아웃 대 개략(LVS)

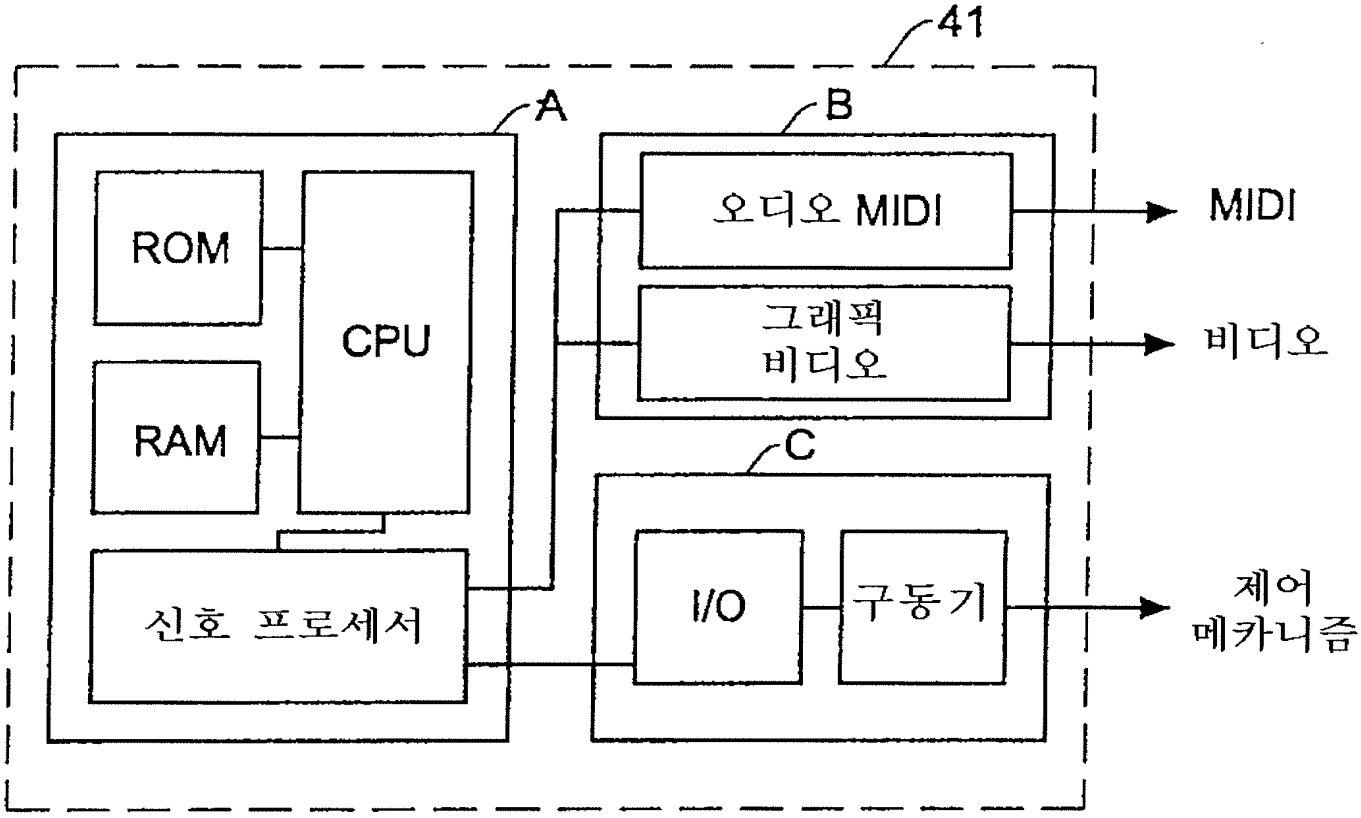
도면 2


S31
코어기능 테스트벤치와 실리콘 IC를 사용하여 개별 코어들을 검증한다
S32
글루로직용 FPGA/에뮬레이션을 사용하여 인터페이스, 온-칩 버스 및 글루 로직을 검증한다
S33
코어-투-코어 통신용 시뮬레이션 테스트벤치와 SoC 레벨 임계 경로들을 사용하여 타이밍을 검증한다
S34
전체 기능 시뮬레이션 테스트벤치 및 어플리케이션 실행을 사용하여 전체 설계 타당성을 검증한다

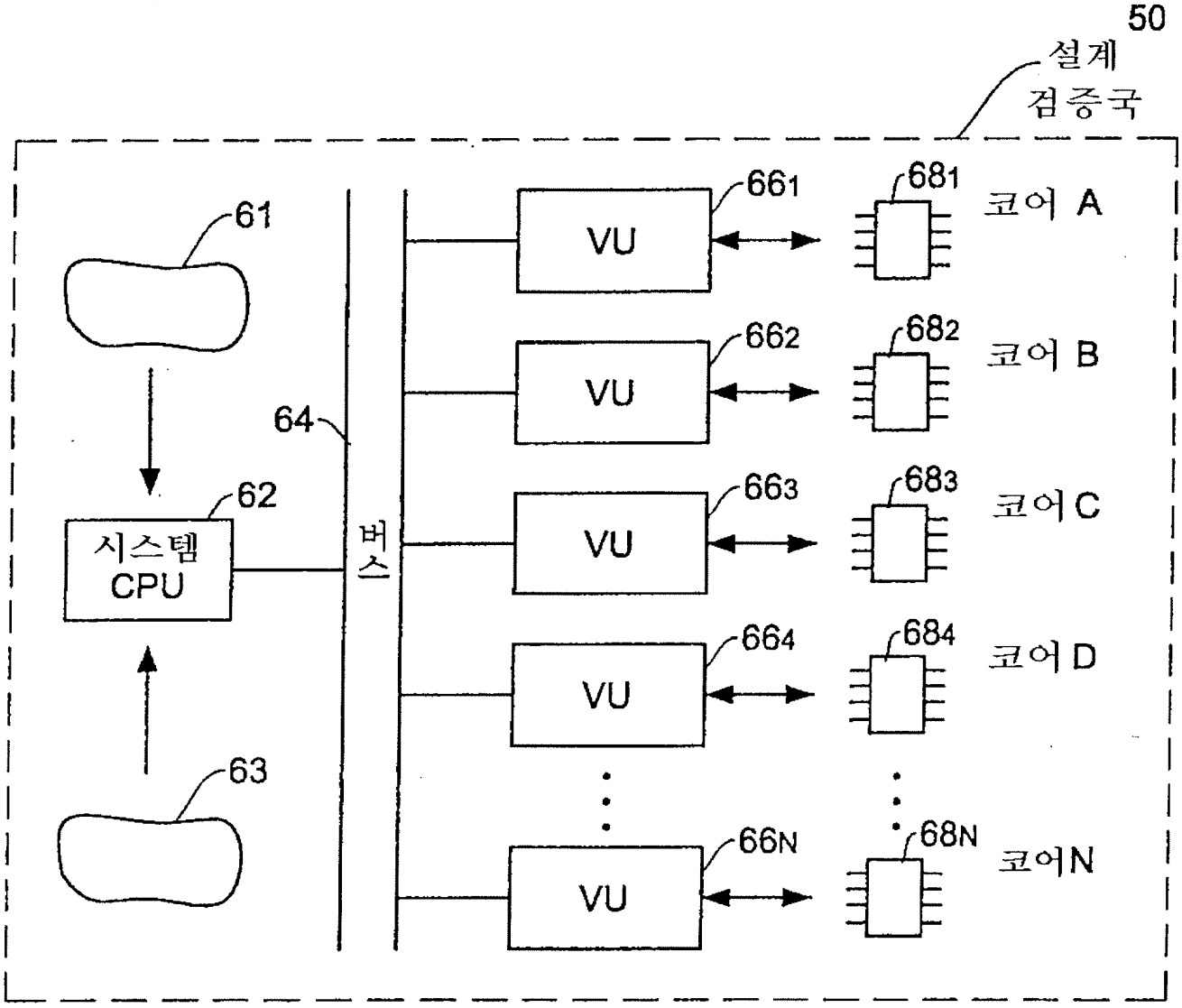
도면 3


41 EDA
43
A
B
C
SOC
신호
입출력
인터페이스
제어
45
설계 데이터
46
실리콘
프로세스
49
DUT
53
테스트 데이터
54
테스트벤치
50 설계검증국
55
VCD
58
이벤트 파일
59
이벤트
테스터

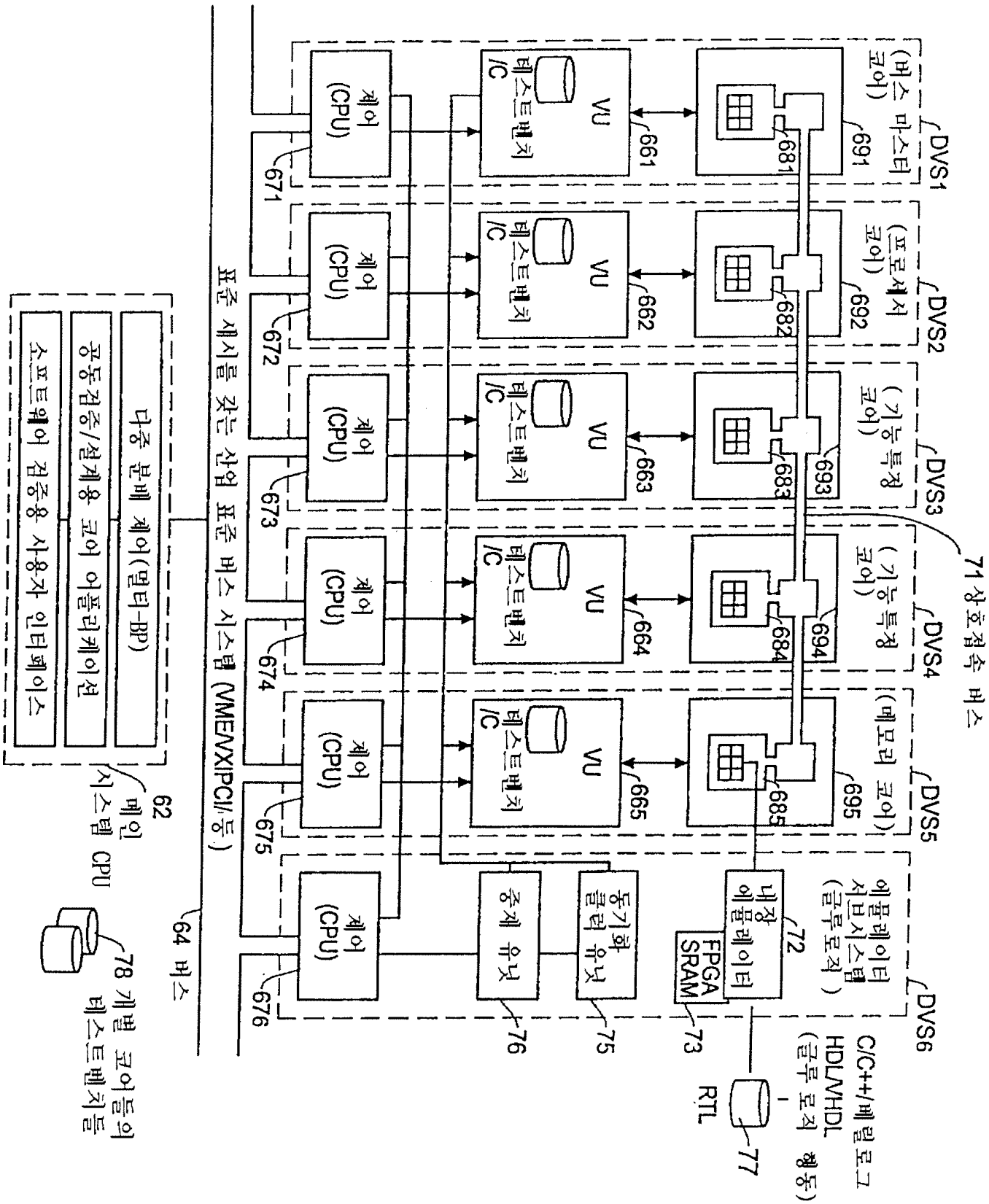
도면 4a


41
A
B
C
ROM
CPU
RAM
신호 프로세서
오디오 MIDI
그래픽
비디오
I/O
구동기
MIDI
비디오
제어
메카니즘

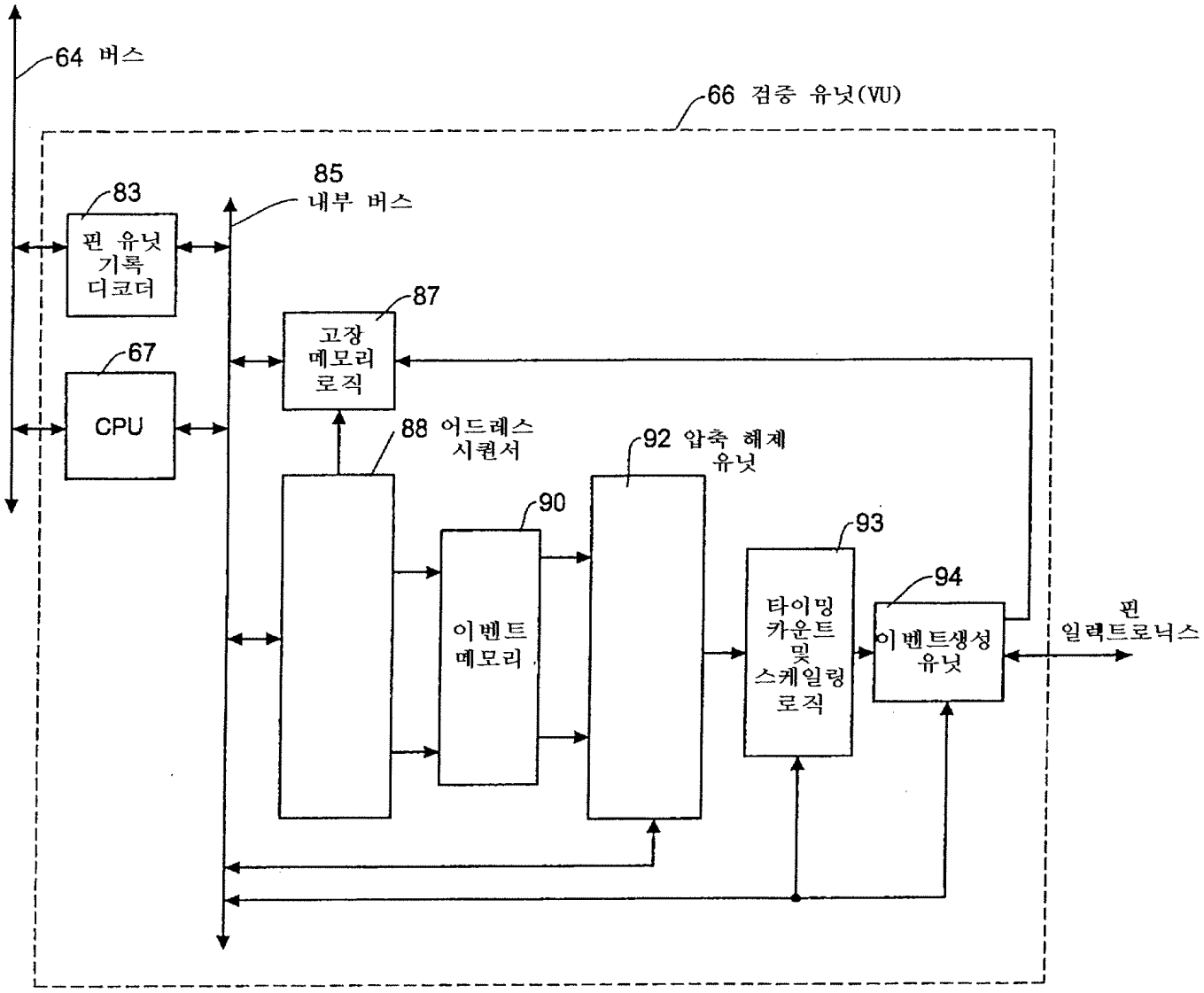
도면 4b


50
설계
검증국
61
62
시스템
CPU
63
64
버스
VU
661
662
663
664
66N
681
682
683
684
68N
코어 A
코어 B
코어 C
코어 D
코어N

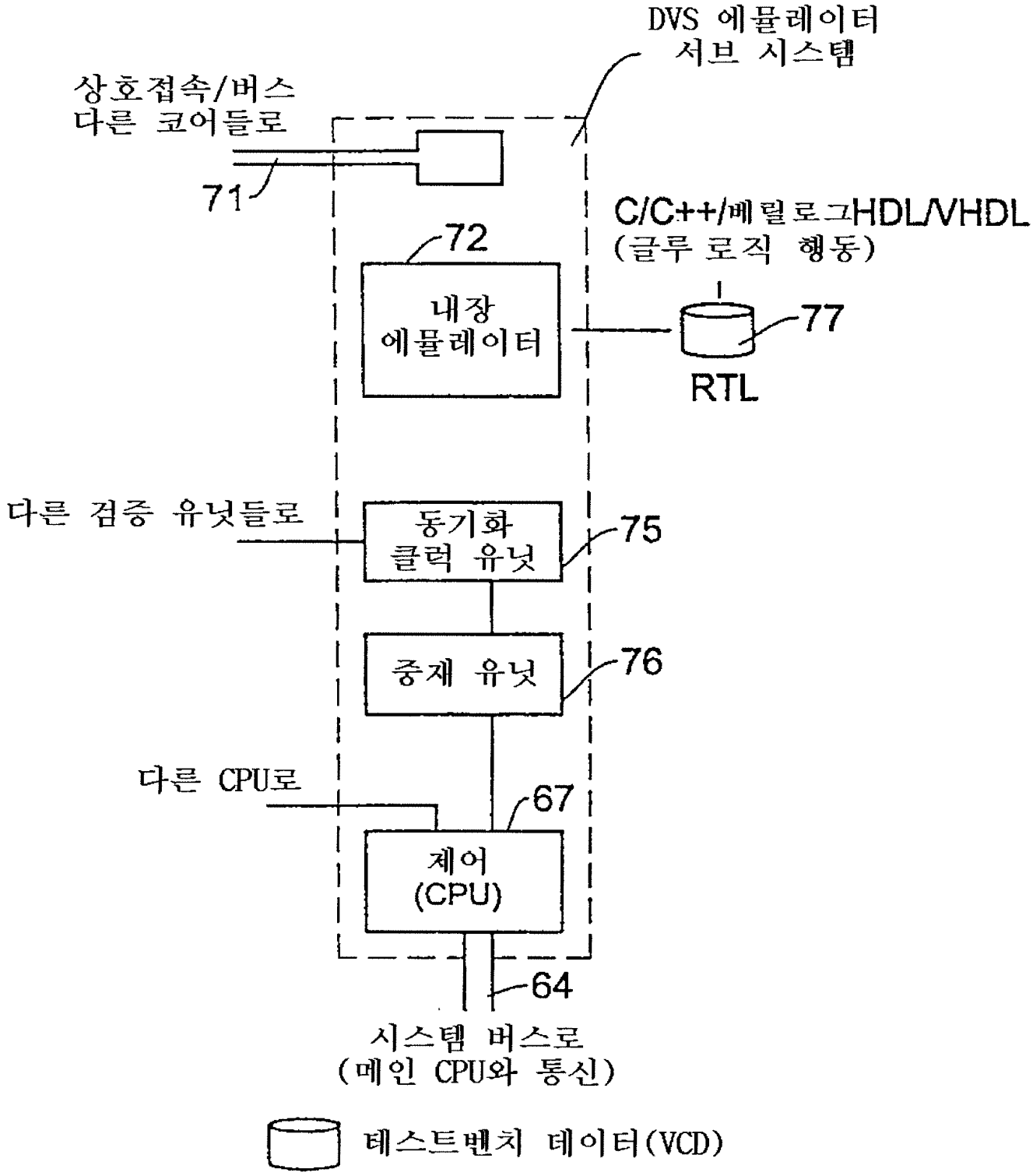
도면 5


71 상호접속 버스
DVS1
(버스 마스터 코어)
691
DVS2
(프로세서 코어)
692
DVS3
(기능특정 코어)
693
DVS4
(기능특정 코어)
694
DVS5
(메모리 코어)
695
DVS6
에뮬레이터 서브시스템 (글루로직)
72
681
682
683
684
685
661
662
663
664
665
671
672
673
674
675
676
VU
테스트벤치 I/C
제어 (CPU)
내장 에뮬레이터
FPGA SRAM
73
RTL
77
C/C++/베릴로그 HDL/VHDL (글루로직 행동)
동기화 클럭 유닛
75
중재 유닛
76
표준 세시를 갖는 산업 표준 버스 시스템 (VME/VXIPCI/등)
64 버스
다중 분배 제어(멀티-BP)
공동검증/설계용 코어 어플리케이션
소프트웨어 검증용 사용자 인터페이스
62
메인 시스템 CPU
78 개별 코어들의 테스트벤치들

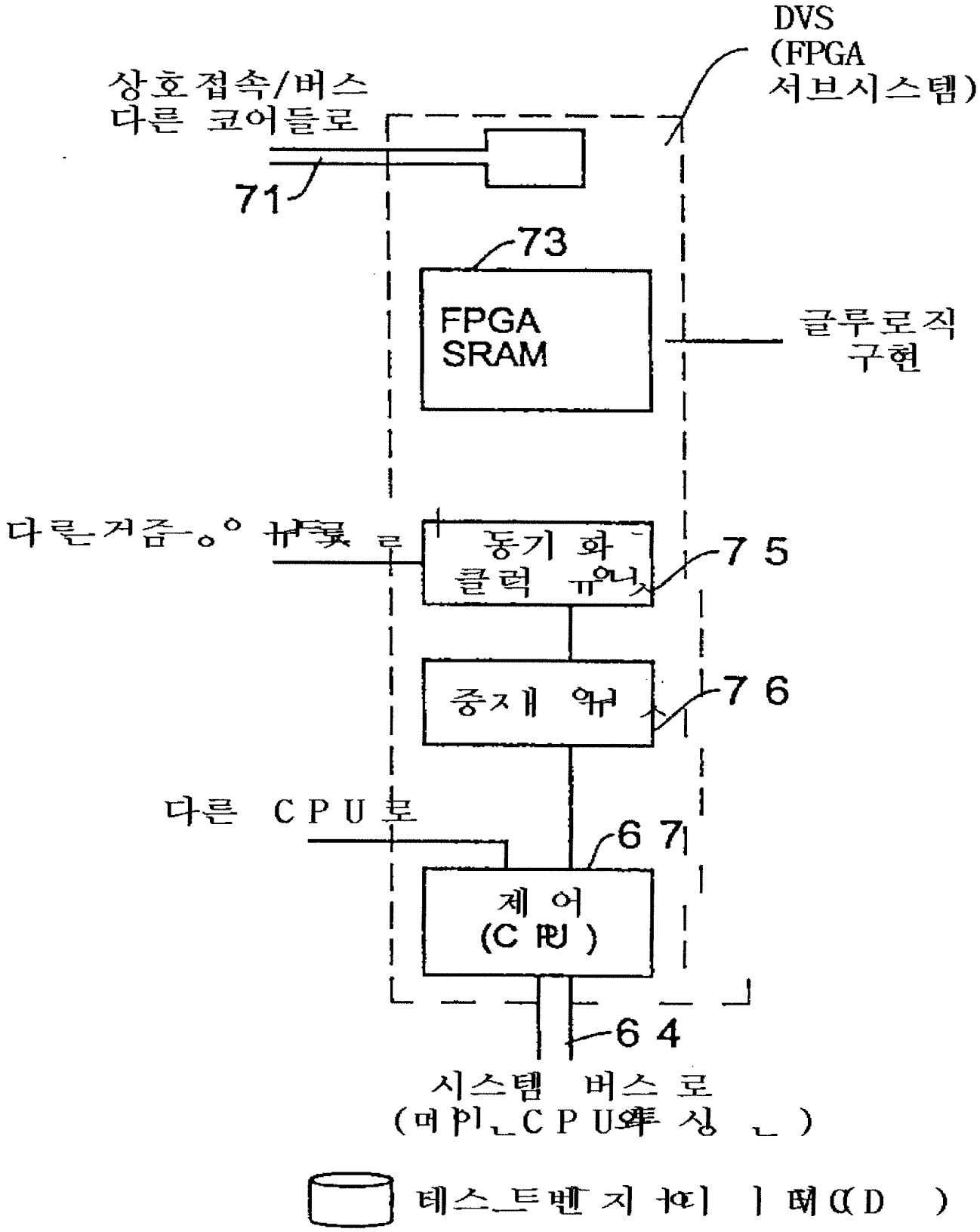
도면 6


64 버스
66 검증 유닛(VU)
85
내부 버스
83
핀 유닛
기록
디코더
87
고장
메모리
로직
67
CPU
88 어드레스
시퀀서
92 압축 해제
유닛
90
이벤트
메모리
93
타이밍
카운트
및
스케일링
로직
94
이벤트생성
유닛
핀
일렉트로닉스

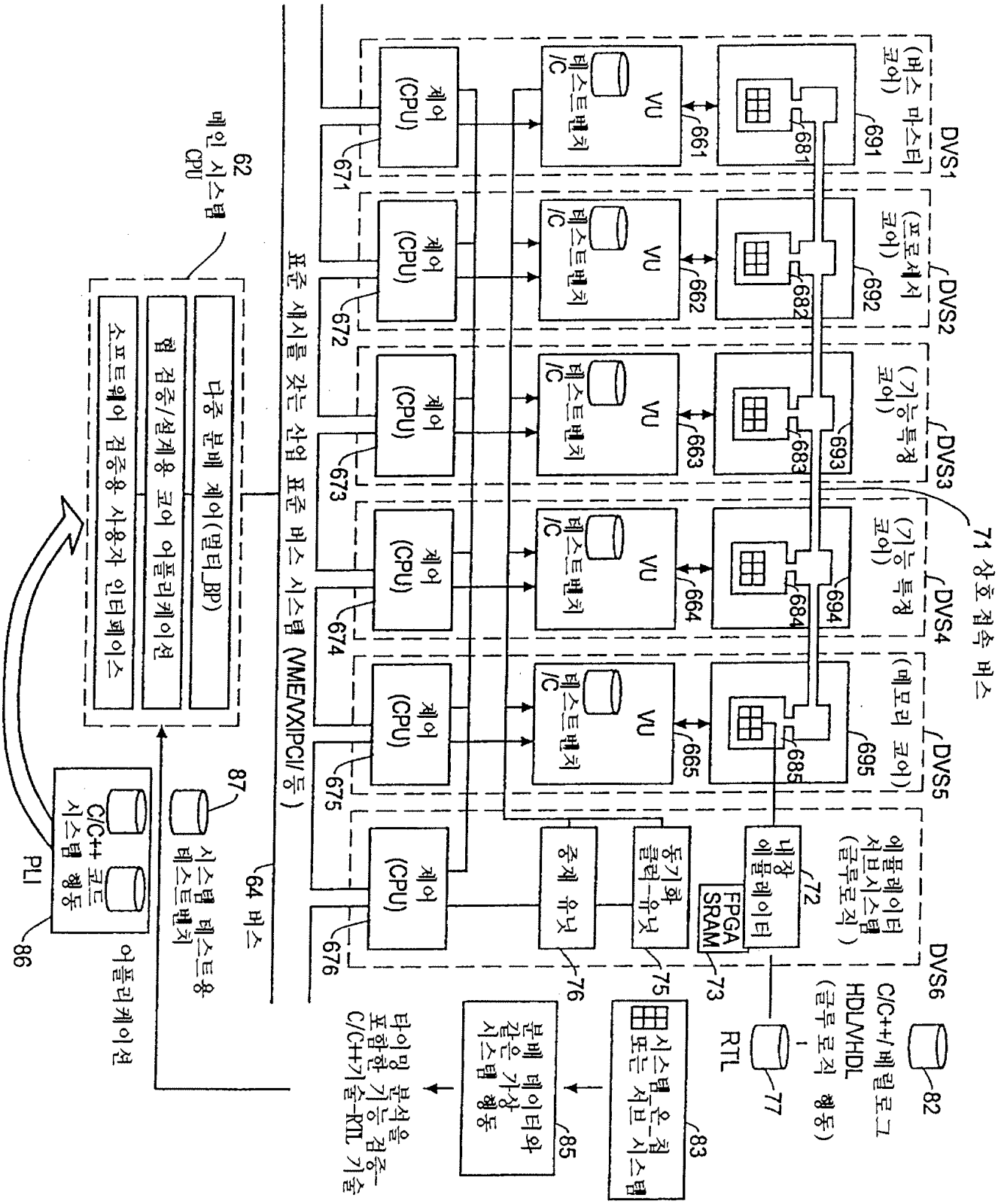
도면 7


DVS 에뮬레이터
서브 시스템
상호접속/버스
다른 코어들로
71
C/C++/베릴로그HDL/VHDL
(글루 로직 행동)
72
내장
에뮬레이터
77
RTL
다른 검증 유닛들로
동기화
클럭 유닛
75
중재 유닛
76
다른 CPU로
67
제어
(CPU)
64
시스템 버스로
(메인 CPU와 통신)
테스트벤치 데이터(VCD)

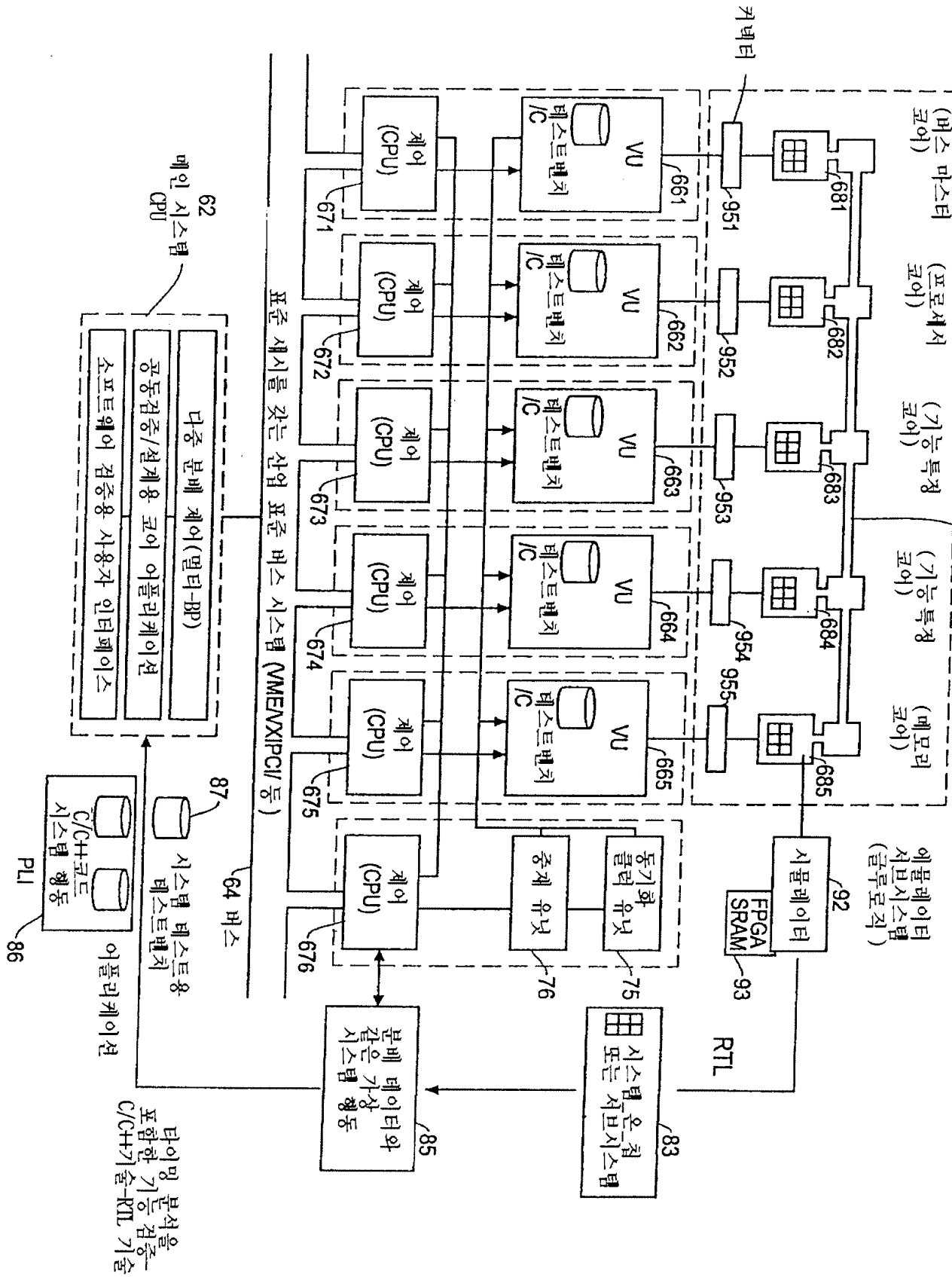
도면 8


DVS
(FPGA
서브시스템)
상호접속/버스
다른 코어들로
71
73
FPGA
SRAM
글루로직
구현
다른거즘ㅇ°
동기화
클럭
75
중지
76
다른 CPU로
67
제어
(CPU)
64
시스템 버스로
테스트벤지
데이터

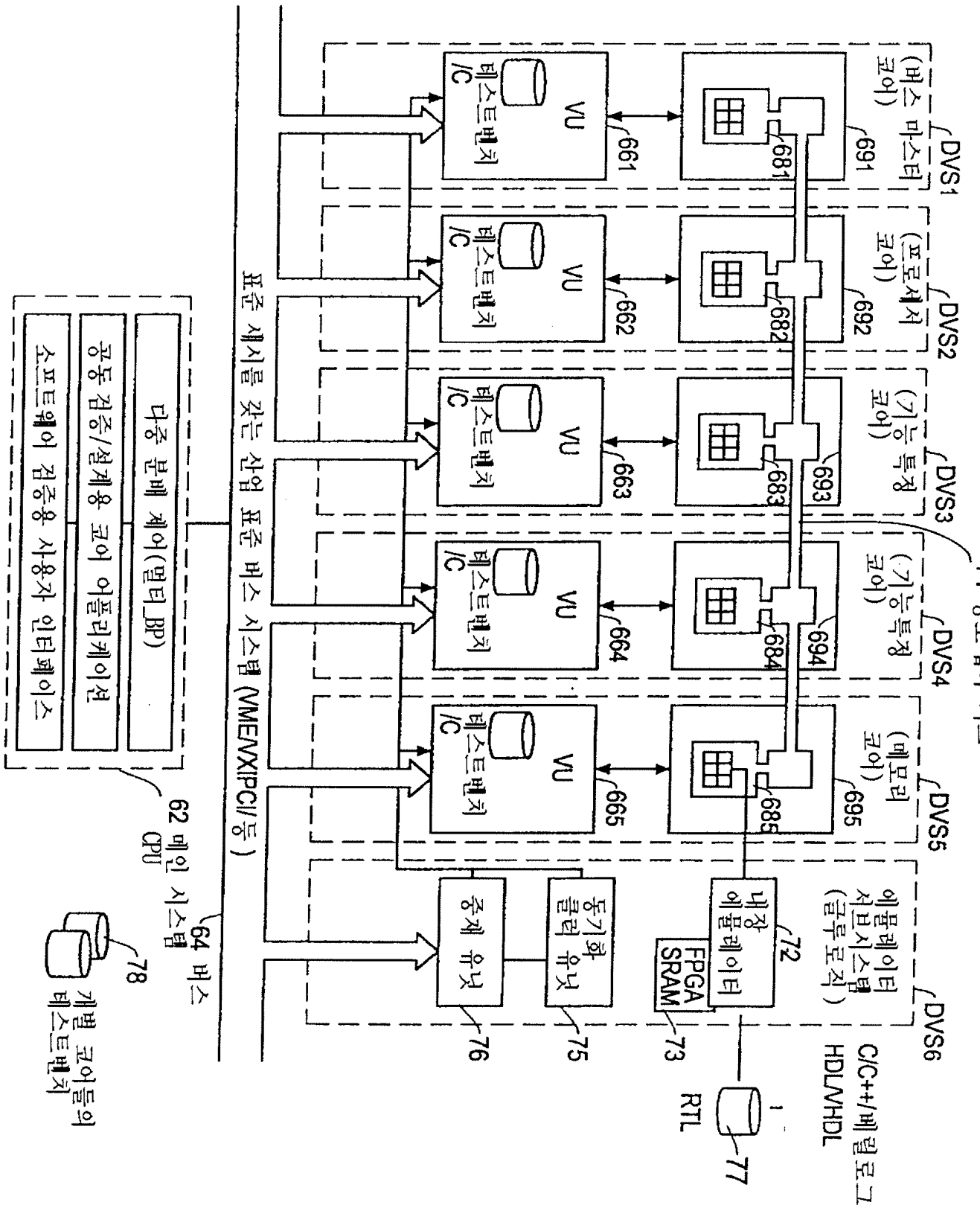
도면 9


DVS1
(버스 마스터 코어)
691
DVS2
(프로세서 코어)
692
DVS3
(기능특정 코어)
693
DVS4
(기능 특정 코어)
694
DVS5
(메모리 코어)
695
DVS6
71 상호 접속 버스
681
682
683
684
685
661
662
663
664
665
VU
테스트벤치 /C
671
672
673
674
675
676
제어 (CPU)
표준 새시를 갖는 산업 표준 버스 시스템 (VME/VXI/PCI/등)
64 버스
62
메인 시스템 CPU
다중 분배 제어(멀티_BP)
협 검증/설계용 코어 어플리케이션
소프트웨어 검증용 사용자 인터페이스
87
시스템 테스트용 테스트벤치
86
C/C++ 코드 시스템 행동
PLI
어플리케이션
에뮬레이터 처브시스템 (글루로직)
72
내장 에뮬레이터
FPGA SRAM
73
동기화 클럭-유닛
75
중재 유닛
76
C/C++/베릴로그 HDL/VHDL (글루 로직 행동)
82
RTL
77
시스템 온-칩 또는 처브 시스템
83
분배 데이터와 같은 가상 시스템 행동
85
타이밍 분석을 포함한 기능 검증 C/C++기술-RTL 기술

도면 10


90 성능 보드
71 상호 접속 버스
(버스 마스터 코어)
(프로세서 코어)
(기능 특정 코어)
(기능특정 코어)
(메모리 코어)
커넥터
951
952
953
954
955
681
682
683
684
685
VU
테스트벤치
I/C
661
662
663
664
665
제어 (CPU)
671
672
673
674
675
676
표준 시스템 버스 (VME/XIPCI/ 등)
64 버스
62 메인 시스템 CPU
다중 분배 제어(멀티-BP)
공동검증/설계용 코어 어플리케이션
소프트웨어 검증용 사용자 인터페이스
87
시스템 테스트용 테스트벤치
86
C/C++코드 시스템 행동
PLI
어플리케이션
에뮬레이터 서브시스템 (글루로직)
92
시뮬레이터
FPGA SRAM
93
RTL
동기화 클럭 유닛
중재 유닛
75
76
분배 테이터와 같은 가상 시스템 행동
85
시스템 온 칩 또는 서브시스템
83
타이밍 분석을 포함한 기능 검증 C/C++기술-RTL 기술

도면 11


71 상호 접속 버스
DVS1
(버스 마스터 코어)
691
DVS2
(프로세서 코어)
692
DVS3
(기능 특정 코어)
693
DVS4
(기능 특정 코어)
694
DVS5
(메모리 코어)
695
DVS6
에뮬레이터 서브시스템 (글루 로직)
681
682
683
684
685
661
662
663
664
665
VU
테스트 벤치 /C
표준 시스템 산업 표준 버스 시스템 (VME/XIPCI/등)
64 버스
62 메인 CPU
다중 분배 제어(멀티_BP)
공동 검증/설계용 코어 어플리케이션
소프트웨어 검증용 사용자 인터페이스
78
개별 코어들의 테스트 벤치
72
내장 에뮬레이터
FPGA SRAM
73
77
RTL
C/C++/베릴로그 HDL/VHDL
동기화 클럭 유닛
75
중재 유닛
76